THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Applicant(s):	Giovanni Benini et al.
Appl. No.:	10/006,314
Conf. No.:	9907
Filed:	December 4, 2001
Title:	METHOD FOR USING A DATA PROCESSING SYSTEM AS A FUNCTION OF AN AUTHORIZATION, ASSOCIATED DATA PROCESSING SYSTEM AND ASSOCIATED PROGRAM
Art Unit:	2136
Examiner:	Colin, Carl G.
Docket No.:	112740-360

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

SUPPLEMENTAL APPEAL BRIEF

Sir:

The Appellants submit this Supplemental Appeal Brief in response to the Notification of Non-Compliant Appeal Brief dated December 26, 2006. This brief supplements the Appeal Brief filed November 21, 2006, filed in support of the Notice of Appeal filed on July 21, 2006. This Appeal is taken from the Advisory Action dated July 10, 2006 and the Final Rejection dated March 21, 2006, which are attached as Appendix A and Appendix B.

I. REAL PARTY IN INTEREST

The real party in interest for the above-identified patent application on appeal is Siemens Aktiengesellschaft by virtue of an Assignment recorded at the United States Patent and Trademark Office on March 25, 2002, at reel 012751, frame 0067.

II. RELATED APPEALS AND INTERFERENCES

Appellants, Appellant's legal representative and the Assignee of the above-identified patent application do not know of any prior or pending appeals, interferences or judicial proceedings which may be related to, directly affect or be directly affected by or have a bearing on the Board's decision with respect to the above-identified Appeal.

III. STATUS OF CLAIMS

Claims 1-12 are pending in the above-identified patent application, with claims 1, 11 and 12 being the only independent claims. As indicated in the Advisory Action, the previous §101 rejection to claim 12 has been withdrawn. However, claims 1-12 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Non-Patent Literature entitled "A Language for User Control of Internet Telephony Services," to *Lennox* et al. in view of U.S. Patent No. 6,476,833 to *Moshfeghi*. Accordingly, claims 1-12 are being appealed in this Brief. A copy of the appealed claims is attached in the Claims Appendix. Copies of the Lennox and Moshfeghi references are attached in Appendix C and D, respectively.

IV. STATUS OF AMENDMENTS

Independent claims 1, 11 and 12 were amended in the Response to the final Office Action dated March 21, 2006. In the Advisory Action dated July 10, 2006, the Examiner indicated that the amendments made to overcome the §101 rejections were entered, but the amendments made to overcome the §103 rejections were not. In particular, the Examiner indicated that the claim amendments proposed for overcoming the §103 rejections raised new issues requiring further search and consideration. No other substantive amendments were made in the application after the final rejection.

V. SUMMARY OF CLAIMED SUBJECT MATTER

The present invention as recited in independent claims 1, 11 and 12 is directed to a system, method and computer program for using a data processing system as a function of authorization in Internet Telephony.

Independent claims 1 and 12 recite a method and computer program (FIG. 3A-B) for using a data processing system (10) as a function of an authorization in Internet Telephony (page 1, lines 7-14). A basic authorization level (16) is defined relating to execution of specific instructions using the data processing system for at least one basic user of the data processing system (page 6, lines 15-28). A priority authorization level (18) is defined, which permits execution of instructions with wider ranging access rights in comparison to the instructions of the basic authorization level, for at least one priority user of the data processing system (page 6, lines 15-28; FIG. 2, page 7, lines 10-26). At least one of the instructions and a syntax of the instructions for the basic authorization level are noted in a basic file section (page 6, lines 15-28; page 7, lines 18-26). At least one of the instructions and a syntax of the instructions for the priority authorization level is noted in a priority file section determining the authorization level of a user before the execution of the instructions of the user (page 6, lines 15-28; page 7, lines 18-26; page 8, lines 13-15). Using one of the basic file section and the priority file section, as a function of the authorization levels determined, the instructions which the user can execute is defined (page 8, lines 16-25).

Independent claim 11 recites a data processing system (FIG. 1, 10) which is used as a function of an authorization in Internet Telephony (page 1, lines 7-14). The system includes a part (16) for defining a basic authorization level relating to execution of specific instructions using the data processing system for at least one basic user of the data processing system (page 6, lines 15-28; see also code on page 9, line 18 - page 10, line 1). Also, a part (18) is recited for defining a priority authorization level, which permits execution of instructions with wider ranging access rights in comparison to the instructions of the basic authorization level, for at least one priority user of the data processing system (page 6, lines 15-28; FIG. 2, page 7, lines 10-26; see also code page 10, line 14 - end of page). Furthermore, a part (12) is recited for noting at least one of the instructions and a syntax of the instructions for the basic authorization

level in a basic file section, as well as a part (14) for noting at least one of the instructions and a syntax of the instructions for the priority authorization level in a priority file section (page 6, lines 15-28; page 7, lines 18-26; page 8, lines 13-15; see also code page 11). Moreover, a part (12) is recited for determining the authorization level of a user before the execution of the instructions of the user, and a part for using one of the basic file section and the priority file section, as a function of the authorization level determined, to define the instructions which the user can execute (page 8, lines 16-25).

Although specification citations are given in accordance with C.F.R. 1.192(c), these reference numerals and citations are merely examples of where support may be found in the specification for the terms used in this section of the Brief. There is no intention to suggest in any way that the terms of the claims are limited to the examples in the specification. Pointing out specification support for the claim terminology as is done here to comply with rule 1.192(c) does not in any way limit the scope of the claims to those examples from which they find support. Nor does this exercise provide a mechanism for circumventing the law precluding reading limitations into the claims from the specification. In short, the references numerals and specification citations are not to be construed as claim limitations or in any way used to limit the scope of the claims.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Claims 1-12 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Non-Patent Literature entitled "A language for User Control of Internet Telephony Services," to *Lennox* et al in view of U.S. Patent No. 6,476,833 to *Moshfeghi*.

VII. ARGUMENT

A. LEGAL STANDARDS

1. Obviousness under 35 U.S.C. § 103

In making a determination that an invention is obvious, the Patent Office has the initial burden of establishing a *prima facie* case of obviousness. *In re Rijckaert*, 9 F.3d 1531, 1532, 28 U.S. P.Q.2d 1955, 1956 (Fed. Cir. 1993). "If the examination at the initial stage does not produce a *prima facie* case of unpatentability, then without more the applicant is entitled to grant of the patent." *In re Oetiker*, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992).

In order to maintain a *prima facie* case of obviousness under 35 U.S.C. §103, the Examiner must satisfy the following criteria: 1) a suggestion or motivation, either in the cited references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the references or combine their teachings to arrive at the invention; 2) a reasonable expectation of success at arriving at the invention, if the combination of the cited references is made; and 3) a teaching of suggestion of all the recited claim limitations in the combination of the cited references. (see MPEP 2142).

The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991). The initial burden is on the examiner to provide some suggestion of the desirability of doing what the inventor has done. "To support the conclusion that the claimed invention is directed to obvious subject matter, either the references must expressly or impliedly suggest the claimed invention or the examiner must present a convincing line of reasoning as to why the artisan would have found the claimed invention to have been obvious in light of the teachings of the references." *Ex parte Clapp*, 227 USPQ 972, 973 (Bd. Pat. App. & Inter. 1985). When the motivation to combine the teachings of the references is not immediately apparent, it is the duty of the examiner to explain why the combination of the teachings is proper. *Ex parte Skinner*, 2 USPQ2d 1788 (Bd. Pat. App. & Inter. 1986). (see MPEP 2142).

Further, the Federal Circuit has held that it is "impermissible to use the claimed invention as an instruction manual or 'template' to piece together the teachings of the prior art so that the claimed invention is rendered obvious." *In re Fritch*, 23 U.S.P.Q.2d 1780, 1784 (Fed. Cir.

1992). "One cannot use hindsight reconstruction to pick and choose among isolated disclosures in the prior art to deprecate the claimed invention" *In re Fine*, 837 F.2d 1071 (Fed. Cir. 1988).

Moreover, the Federal Circuit has held that "obvious to try" is not the proper standard under 35 U.S.C. §103. *Ex parte Goldgaber*, 41 U.S.P.Q.2d 1172, 1177 (Fed. Cir. 1996). "An-obvious-to-try situation exists when a general disclosure may pique the scientist curiosity, such that further investigation might be done as a result of the disclosure, but the disclosure itself does not contain a sufficient teaching of how to obtain the desired result, or that the claim result would be obtained if certain directions were pursued." *In re Eli Lilly and Co.*, 14 U.S.P.Q.2d 1741, 1743 (Fed. Cir. 1990).

B. THE REJECTION OF CLAIMS 1, 11 AND 12 UNDER 35 U.S.C. §103(A) IS IMPROPER BECAUSE *LENNOX ET AL.* IN VIEW OF *MOSHFEGHI* DOES NOT RENDER OBVIOUS THE CLAIMED INVENTION

1. *Lennox et al.* and *Moshfeghi*, alone or in combination, fail to disclose or suggest all of the elements of the claimed invention

Claims 1-12 stand rejected under 35 U.S.C. §103(a) as being unpatentable over *Lennox et al.* ("A Language for User Control of Internet Telephony Services) in view of *Moshfeghi* (US Patent 6,467,833).

The cited art, alone or in combination, fails to teach or suggest the feature of "defining a priority authorization level, which permits execution of instructions with wider ranging access rights in comparison to the instructions of the basic authorization level, for at least one priority user of the data processing system" as recited in claim 1 and similarly recited in claims 11 and 12.

Regarding *Lennox*, the disclosure teaches various CPL script arrangements for call processing (see sections 2.1-2.3, pages 2-4). In the passages cited by the Office Action (sections 6.1-7.2, pages 20-27), the disclosure discusses a CPL location model (see also section 2.3, page 4), where signaling operations are dependent upon location sets, where location nodes may be added or removed. Under the disclosure, users may be authorized to access specific nodes, depending on whether or not they are in a specified location (section 6.1). Furthermore, authorization is conducted during connection to protect users against denial of service attacks (FIGS. 28 and 29; section 14).

However, nowhere in the disclosure of *Lennox* is it disclosed that *a priority authorization level is used to permit execution of instructions with wider ranging access rights.* In fact, the proxy mode of page 25 has nothing to do with authorization. Instead, page 25 merely describes prioritizing a list of alternate locations to which a call may be forwarded. Additionally, *Lennox* at pages 40 and 43 merely describe a call set-up, and the code illustrated in FIG. 29 merely authenticates a user's SIP to a voicemail command. Finally, *Lennox* states that "a method by which scripts are transmitted from a client to a server must be strongly authenticated." However,

"[s]uch a method is not specified in this document." (see, page 42). Thus, none of the cited sections of *Lennox* relate to prioritized authorization of access rights, as claimed.

Furthermore, *Moshfeghi* does not solve the deficiencies of *Lennox*, discussed above. The broadly cited passage of col. 2, line 55 – col. 4, line 32 teaches the use of embedded browser functions to prevent unauthorized linking (col. 3, lines 8-13; 26-30) and is completely silent with regard to prioritized authorization.

2. One having ordinary skill in the art would not be motivated to combine *Lennox et al.* in view of *Moshfeghi* to arrive at the present claims

Technology disclosed in *Moshfeghi* is directed to browser-based TCP/IP linking over a conventional computer network, and has nothing whatsoever to do with the CPL scripting disclosed in *Lennox*. *Lennox* expressly deals with non-URL sources (page 22, 2nd-4th paragraph), where caller preferences and capabilities are used as alternate parameters (page 22, 5th-7th paragraph) to allow server resolution of addresses to be added to a location set. None of this is remotely addressed in *Moshfeghi*.

C. THE PATENTABILITY OF CLAIM 1 RENDERS MOOT THE REJECTIONS OF CLAIMS 2-10

Dependent claims 2-10 were also rejected under 35 U.S.C. §103(a) as being unpatentable over *Lennox et al.* in view of *Moshfeghi*. Appellants respectfully submit that the patentability of independent claim 1 as previously discussed renders moot the obviousness rejections of claims 2-10. In this regard, the cited art fails to teach or suggest the elements of these claims in direct/indirect combination with their respective independent claims.

VIII. CONCLUSION

The Appellants respectfully submit that claims 1-12 are non-obvious in view of the cited art. Accordingly, the Appellants respectfully submit that the rejections of pending claims 1-12 are erroneous in law and fact and should be reversed by this Board.

The Director is authorized to charge any fees which may be required to Deposit Account No. 02-1818. If such a withdrawal is made, please indicate the Attorney Docket No. 112740-360 on the account statement.

Respectfully submitted,

BELL, BOYD & LLOYD LLC

BY _____

Peter Zura
Reg. No. 48,196
Customer No.: 29177
Phone: (312) 807-4208

Dated: January 26, 2007

CALIMS APPENDIX

PENDING CLAIMS ON APPEAL OF U.S. PATENT APPLICATION SERIAL NO. 10/006,314

Listing of Claims:

Claim 1. (previously presented) A method for using a data processing system as a function of an authorization in Internet Telephony, the method comprising the steps of:

defining a basic authorization level relating to execution of specific instructions using the data processing system for at least one basic user of the data processing system;

defining a priority authorization level, which permits execution of instructions with wider ranging access rights in comparison to the instructions of the basic authorization level, for at least one priority user of the data processing system;

noting at least one of the instructions and a syntax of the instructions for the basic authorization level in a basic file section;

noting at least one of the instructions and a syntax of the instructions for the priority authorization level in a priority file section;

determining the authorization level of a user before the execution of the instructions of the user; and

using one of the basic file section and the priority file section, as a function of the authorization levels determined, to define the instructions which the user can execute.

Claim 2. (original) A method for using a data processing system as a function of an authorization as claimed in claim 1, the method further comprising the steps of:

storing the basic file section in a basic file; and

storing the preferred file section in a priority file, which differs from the basic file.

Claim 3. (original) A method for using a data processing system as a function of an authorization as claimed in claim 1, wherein at least one of the basic file section and the priority file section does not itself define a program or program section which can be executed by a processor.

Claim 4. (original) A method for using a data processing system as a function of an authorization as claimed in claim 1, the method further comprising the step of:

defining the instructions of the basic authorization level and at least one of an additional instruction and an expanded syntax in comparison with the syntax of the basic authorization level for the priority authorization level.

Claim 5. (original) A method for using a data processing system as a function of an authorization as claimed in claim 1, the method further comprising the steps of:

transmitting, by a user, an instruction file with instructions to the data processing system for determining the authorization level;

checking the instructions contained in the instruction file as a function of the authorization level using one of the basic file section and the priority file section; and

storing the instruction file for a later execution if it contains only instructions which are valid for the authorization level which is determined.

Claim 6. (original) A method for using a data processing system as a function of an authorization as claimed in claim 5, the method further comprising the steps of:

determining the authorization level of the user before the processing of the instruction file; and

using one of the basic file section and the priority file section to process the instruction file as a function of the authorization level for the processing of the instruction file.

Claim 7. (original) A method for using a data processing system as a function of an authorization as claimed in claim 5, wherein the basic file section and the priority file section contain at least one of instructions and a syntax of the instructions of a markup language, which is used to described contents of character chains, the markup language being selected from the group consisting of SGML, XML, HTML 4.0, and a markup language based on one of these languages, such that the instruction file contains instructions in the markup language.

Claim 8. (original) A method for using a data processing system as a function of an authorization as claimed in claim 5, wherein the basic file section in the priority file section define at least one of instructions and a syntax of the instructions for controlling a voice transmission via at least one of a circuit-switched telephone network and a packet-switched data transmission network, the syntax of instructions of a language selected from a group consisting of CPL and a language based on CPL, such that the instruction filed defines instructions for controlling the voice transmission.

Claim 9. (original) A method for using a data processing system as a function of an authorization as claimed in claim 5, wherein, for processing the instruction file, a same parser program is used for decomposing the instruction file into individual instructions.

Claim 10. (original) A method for using a data processing system as a function of an authorization as claimed in claim 5, wherein a same application program is used for executing the instructions, irrespective of the authorization level.

Claim 11. (previously presented) A data processing system which is used as a function of an authorization in Internet Telephony, comprising:

a part for defining a basic authorization level relating to execution of specific instructions using the data processing system for at least one basic user of the data processing system;

a part for defining a priority authorization level, which permits execution of instructions with wider ranging access rights in comparison to the instructions of the basic authorization level, for at least one priority user of the data processing system;

a part for noting at least one of the instructions and a syntax of the instructions for the basic authorization level in a basic file section;

a part for noting at least one of the instructions and a syntax of the instructions for the priority authorization level in a priority file section;

a part for determining the authorization level of a user before the execution of the instructions of the user; and

a part for using one of the basic file section and the priority file section, as a function of the authorization level determined, to define the instructions which the user can execute.

Claim 12. (previously presented) A program having executable code stored on a computer-readable medium that when executed by one or more processors performs a method for using a data processing system as a function of an authorization in Internet Telephony, the method comprising:

defining a basic authorization level relating to execution of specific instructions using the data processing system for at least one basic user of the data processing system;

defining a priority authorization level, which permits execution of instructions with wider ranging access rights in comparison to the instructions of the basic authorization level, for at least one priority user of the data processing system;

noting at least one of the instructions and a syntax of the instructions for the basic authorization level in a basic file section;

noting at least one of the instructions and a syntax of the instructions for the priority authorization level in a priority file section;

determining the authorization level of a user before the execution of the instructions of the user; and

using one of the basic file section and the priority file section, as a function of the authorization levels determined, to define the instructions which the user can execute.

EVIDENCE APPENDIX

EXHIBIT A: Advisory Action Mailed on July 10, 2006

EXHIBIT B: Final Office Action Mailed on March 21, 2006

EXHIBIT C: Non-Patent Literature entitled "A Language for User Control of Internet Telephony Services" (*Lennox et al.*)

EXHIBIT D: U.S. Patent No. 6,476,833 ("Moshfeghi.")

RELATED PROCEEDINGS APPENDIX

None

APPENDIX A

Advisory Action Mailed on July 10, 2006

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/006,314	12/04/2001	Giovanni Benini	112740-360	9907

29177 7590 07/10/2006

BELL, BOYD & LLOYD, LLC
P. O. BOX 1135
CHICAGO, IL 60690-1135

EXAMINER	
COLIN, CARL G	

ART UNIT	PAPER NUMBER
2136	

DATE MAILED: 07/10/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Advisory Action Before the Filing of an Appeal Brief	**Application No.** 10/006,314	**Applicant(s)** BENINI, GIOVANNI	
	Examiner Carl Colin	**Art Unit** 2136	

--The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

THE REPLY FILED 20 June 2006 FAILS TO PLACE THIS APPLICATION IN CONDITION FOR ALLOWANCE.

1. ☒ The reply was filed after a final rejection, but prior to or on the same day as filing a Notice of Appeal. To avoid abandonment of this application, applicant must timely file one of the following replies: (1) an amendment, affidavit, or other evidence, which places the application in condition for allowance; (2) a Notice of Appeal (with appeal fee) in compliance with 37 CFR 41.31; or (3) a Request for Continued Examination (RCE) in compliance with 37 CFR 1.114. The reply must be filed within one of the following time periods:
   a) ☒ The period for reply expires 3 months from the mailing date of the final rejection.
   b) ☐ The period for reply expires on: (1) the mailing date of this Advisory Action, or (2) the date set forth in the final rejection, whichever is later. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of the final rejection.

   Examiner Note: If box 1 is checked, check either box (a) or (b). ONLY CHECK BOX (b) WHEN THE FIRST REPLY WAS FILED WITHIN TWO MONTHS OF THE FINAL REJECTION. See MPEP 706.07(f).

Extensions of time may be obtained under 37 CFR 1.136(a). The date on which the petition under 37 CFR 1.136(a) and the appropriate extension fee have been filed is the date for purposes of determining the period of extension and the corresponding amount of the fee. The appropriate extension fee under 37 CFR 1.17(a) is calculated from: (1) the expiration date of the shortened statutory period for reply originally set in the final Office action; or (2) as set forth in (b) above, if checked. Any reply received by the Office later than three months after the mailing date of the final rejection, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

NOTICE OF APPEAL

2. ☐ The Notice of Appeal was filed on _____. A brief in compliance with 37 CFR 41.37 must be filed within two months of the date of filing the Notice of Appeal (37 CFR 41.37(a)), or any extension thereof (37 CFR 41.37(e)), to avoid dismissal of the appeal. Since a Notice of Appeal has been filed, any reply must be filed within the time period set forth in 37 CFR 41.37(a).

AMENDMENTS

3. ☒ The proposed amendment(s) filed after a final rejection, but prior to the date of filing a brief, will not be entered because
   (a) ☒ They raise new issues that would require further consideration and/or search (see NOTE below);
   (b) ☒ They raise the issue of new matter (see NOTE below);
   (c) ☐ They are not deemed to place the application in better form for appeal by materially reducing or simplifying the issues for appeal; and/or
   (d) ☐ They present additional claims without canceling a corresponding number of finally rejected claims.

   NOTE: *See Continuation Sheet*. (See 37 CFR 1.116 and 41.33(a)).
4. ☐ The amendments are not in compliance with 37 CFR 1.121. See attached Notice of Non-Compliant Amendment (PTOL-324).
5. ☒ Applicant's reply has overcome the following rejection(s): 101 rejection of claim 12.
6. ☐ Newly proposed or amended claim(s) _____ would be allowable if submitted in a separate, timely filed amendment canceling the non-allowable claim(s).
7. ☒ For purposes of appeal, the proposed amendment(s): a) ☐ will not be entered, or b) ☐ will be entered and an explanation of how the new or amended claims would be rejected is provided below or appended.
   The status of the claim(s) is (or will be) as follows:
   Claim(s) allowed: _____.
   Claim(s) objected to: _____.
   Claim(s) rejected: *1-12*.
   Claim(s) withdrawn from consideration: _____.

AFFIDAVIT OR OTHER EVIDENCE

8. ☐ The affidavit or other evidence filed after a final action, but before or on the date of filing a Notice of Appeal will not be entered because applicant failed to provide a showing of good and sufficient reasons why the affidavit or other evidence is necessary and was not earlier presented. See 37 CFR 1.116(e).
9. ☐ The affidavit or other evidence filed after the date of filing a Notice of Appeal, but prior to the date of filing a brief, will not be entered because the affidavit or other evidence failed to overcome all rejections under appeal and/or appellant fails to provide a showing a good and sufficient reasons why it is necessary and was not earlier presented. See 37 CFR 41.33(d)(1).
10. ☐ The affidavit or other evidence is entered. An explanation of the status of the claims after entry is below or attached.

REQUEST FOR RECONSIDERATION/OTHER

11. ☐ The request for reconsideration has been considered but does NOT place the application in condition for allowance because: _____.
12. ☐ Note the attached Information Disclosure Statement(s). (PTO/SB/08 or PTO-1449) Paper No(s). _____
13. ☐ Other: _____.

Continuation of 3. NOTE: Applicant has amended the independent claims to recite that "the specific instructions are received in an Internet Markup Language and parsed user a parser" . Applicant mentioned page 5, lines 5-15 for support. However, the citation provided by Applicant does not refer to parser nor the claimed limitation as amended. The proposed amendment will not be entered because they raise the issue of new matter and therefore, they require at least further consideration. It is noted that the use of Markup Language was already cited in the dependent claims and disclosed by the reference as explained in the Office action.

AYAZ SHEIKH
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

APPENDIX B

Final Office Action Mailed on March 21, 2006.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/006,314	12/04/2001	Giovanni Benini	112740-360	9907

29177 7590 03/21/2006

BELL, BOYD & LLOYD, LLC
P. O. BOX 1135
CHICAGO, IL 60690-1135

EXAMINER	
COLIN, CARL G	

ART UNIT	PAPER NUMBER
2136	

DATE MAILED: 03/21/2006

Due: 6-21-06

References Downloaded

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/006,314	BENINI, GIOVANNI	
	Examiner	Art Unit	
	Carl Colin	2136	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *12/27/2005*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-12* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-12* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *04 December 2001* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☒ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

Response to Arguments

1. In response to communications filed on 12/27/2005, applicant amends claim 12. The following claims 1-12 are presented for examination.

1.1 In response to communications filed on 12/27/2005, Applicant has amended the specification to overcome the objection from the last Office Action. However, the amendment does not reflect any changes.

Applicant's remarks, pages 8-9, filed on 12/27/2005, with respect to the rejection of claims 1-12 have been fully considered but they are not persuasive. Applicant argues that the reference does not teach a priority level with wider ranging access rights. Examiner respectfully disagrees. Lenox does disclose defining CPL extensions (priority authorization level with wider ranging access rights) to the basic or default behavior (basic authorization level), page 25 states "CPL extensions to allow in-call or end-of-call operations will require an additional output, such as success to be added"... users may specify that the one with highest priority be tried first (see example on page 25). Also, in another embodiment as specified in the office action, page 40 provides an example where a priority authorization level is defined by the user which permits call from his boss to be directed to his mobile phone and all other calls to be directed to voicemail. Several examples of CPL extensions that define priority authorization level are disclosed throughout the disclosure. For at least the reasons cited above and in the Office Action applicant has not overcome the prior art and claims 1-12 remain rejected.

Specification

2. The disclosure is objected to because it contains embedded hyperlinks and/or other form of browser-executable codes (see page 2, line 3; page 10, line 5). Applicant is required to delete the embedded hyperlinks and/or other form of browser-executable codes. Applicant is suggested to enclose the hyperlinks in quotation marks, remove the "http://www" or appropriate correction to make them non-executable. See MPEP § 608.01.

Claim Rejections - 35 USC § 101

3. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

Claim 12 is rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. The program recited in claim12 without a computer-readable medium needed to realize the computer program's functionality is non-statutory functional descriptive material. Claim 12 reciting "The program having a command sequence ... comprising"; the claim is directed to a program per se and is not embodied in a computer readable medium, in addition it is not executed by a machine. It appears that the preamble recites a processor executing a method and the claim is claiming a program per se. See MPEP § 2106. IV.B.1(b).

See also "http://www.uspto.gov/web/offices/dcom/bpai/prec/2003-2088.pdf".

```
Appeal No. 2003-2088 Application 08/093,516
claim 6 is not in one of the categories of § 101); In re
Bonczyk,10 Fed. Appx. 908 (Fed. Cir. 2001) (non-
```

precedential)("fabricated energy structure" does not correspond to any statutory category of subject matter). Music, art, and literature, if claimed as such, do not fit into any of the statutory categories because they are not physical things or acts. Another example seen in the USPTO is a claim to a computer program per se, i.e., a claim reciting solely a program comprising a set of computer instructions for performing certain functions, instead of a series of steps performed on a computer. The instructions are not physical things which would qualify as a machine, manufacture, or composition of matter, and the claim is not recited as a series of steps as a process. Thus, a computer program per se is not statutory subject matter because it does not fall within any statutory class. See In re Chatfield,545 F.2d 152, 159, 191 USPQ 730, 737 (CCPA 1976) (Rich, J.,dissenting) ("It has never been otherwise than perfectly clear to those desiring patent protection on inventions which are new and useful programs for general purpose computers (software) that the only way it could be obtained would be to describe and claim(35 U.S.C. § 112) the invention as a 'process' or a 'machine.'");In re Lowry, 32 F.3d 1579, 32 USPQ2d 1031 (Fed. Cir. 1994)(memory containing a stored data structure was statutory subject matter, which has been interpreted to mean that programs stored on a physical medium are statutory subject matter as a "manufacture").

A series of steps which is not tied to a particular machine or apparatus, and which does not transform physical subject matter to a different state or thing, does not meet the statutory

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to

which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4.1 **Claims 1-12** are rejected under 35 U.S.C. 103(a) as being unpatentable over Non-Patent Literature "A Language for User Control of Internet Telephony Services", pages 1-60 to **Lennox et al,** in view of US Patent 6,476,833 to **Moshfeghi**.

4.2 **As per claims 1, 2, 6, 7, 11, and 12, Lennox et al** substantially teaches a method for using a data processing system as a function of an authorization, the method comprising the steps of: defining a CPL for location authorization that specifies specific instructions in the markup language using the data processing system to be executed relating only to location (see sections 6.2-7.1, pages 20-24) that meets the recitation of defining a basic authorization level relating to execution of specific instructions using the data processing system for at least one basic user of the data processing system. In another embodiment, figure 29, page 41 shows a complex example of an authorization level of a user that can be extended with a wider range of access rights as shown in figure 28, page 40 in comparison to the instructions of figure 29, for at least one priority user of the data processing system that meets the recitation of defining a priority authorization level, which permits execution of instructions with wider ranging access rights in comparison to the instructions of the basic authorization level, for at least one priority user of the data processing system. **Lennox et al** further suggests on page 40 that the method by which scripts are transmitted from client to servers must be strongly authenticated and servers should allow server administrators to control the details of what CPL operations are performed that meets the recitation of determining the

authorization level of a user before the execution of the instructions of the user; and discloses each script defining the instructions which the user can execute (figures 26-28 and pages 25, 40, and 43) that meets the recitation of using one of the basic file section and the priority file section, as a function of the authorization levels determined, to define the instructions which the user can execute. **Lennox et al** also suggests determining the authorization level of the user before processing (pages 40 and 43). **Lennox et al** discloses in figures 28 and 29, syntax and instructions for the basic and extended authorization level that meets the recitation of noting at least one of the instructions and a syntax of the instructions for the basic authorization level in a basic file section; noting at least one of the instructions and a syntax of the instructions for the priority authorization level in a priority file section; **Lennox et al** discloses that the files comprise scripts (file sections), it is apparent that the files or PCL are stored in the computer (pages 5-7).

Moshfeghi in an analogous art teaches a method and apparatus for providing configurable markup language such as HTML and XML, that restrict users execution of instruction, the method disclosed storing user profile records, the profile records defining a authorization level relating to each user's execution of specific instructions using the data processing system for at least one basic user of the data processing system (see summary of invention, column 2, line 55 through column 4, line 32). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the teaching of **Lennox et al** to store the scripts into users profile records and use one of the basic file section and the priority file section, as a function of the authorization levels determined, to define the instructions which the user can execute as taught by **Moshfeghi** to provide an additional flash memory in the second encryption sub-module and a CMOS memory coupled with the dual-port memory via the first

bus of the dual-port memory containing the encryption keys as taught in IBM Technical Disclosure Bulletin. One skilled in the art would have been lead to make such a modification because it would provide an easy way to control and filter user instructions by being responsive to the content or function privileges in the user's profiles as suggested by **Moshfeghi** (column 3, line 20 through column 4, line 32).

As per claim 3, Lennox et al discloses extended file or program section in figure 28 that is not defined in figure 29 to be executed by a processor other examples can be found on pages 47 et seq. that meets the recitation of wherein at least one of the basic file section and the priority file section does not itself define a program or program section, which can be executed by a processor (see figures 28 and 29).

As per claim 4, Lennox et al suggests defining the instructions of the basic authorization level and at least one of an additional instruction and an expanded syntax in comparison with the syntax of the basic authorization level for the priority authorization level (pages 51-52).

As per claim 5, the combination of Lennox et al and Moshfeghi discloses transmitting, by a user, an instruction file with instructions to the data processing system for determining the authorization level (**Lennox et al,** pages 25-27); checking the instructions contained in the instruction file as a function of the authorization level using one of the basic file section and the priority file section (**Lennox et al,** pages 25-27 and pages 38-48 that also disclose filtering by way of examples); and storing the instruction file for a later execution if it contains only instructions

which are valid for the authorization level which is determined (**Moshfeghi**, column 16, lines 20-40). Claim 5 is therefore rejected on the same rationale as the rejection of claims 1 and 2).

As per claim 8, Lennox et al suggests using voice transmission as a media type for Internet telephony services (see for example, Applicant's disclosure abstract, lines 1-4).

As per claims 9-10, the combination of Lennox et al and Moshfeghi discloses the limitation of wherein, for processing the instruction file, a same parser program is used for decomposing the instruction file into individual instructions and wherein a same application program is used for executing the instructions, irrespective of the authorization level (see **Lennox et al,** figures 28 and 29).

Conclusion

5. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event,

however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

5.1 The prior art made of record and not relied upon is considered pertinent to applicant's disclosure as the art discloses the use of access level control using markup language and user access control relating to instructions that can be executed by a user.

US Patents: 6,931,532 Davis et al ; 6,317,742 Nagaratman et al ; 5,778,365 Nishiyama 6,859,671 Brown.

5.2 Any inquiry concerning this communication or earlier communications from the examiner should be directed to Carl Colin whose telephone number is 571-272-3862. The examiner can normally be reached on Monday through Thursday, 8:00-6:30 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ayaz Sheikh can be reached on 571-272-3795. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

cc

Carl Colin
Patent Examiner
March 17, 2006

CHRISTOPHER REVAK
PRIMARY EXAMINER
Cel 3/16/06

Notice of References Cited	Application/Control No. 10/006,314	Applicant(s)/Patent Under Reexamination BENINI, GIOVANNI	
	Examiner Carl Colin	Art Unit 2136	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-6,476,833	11-2002	Moshfeghi, Mehran	715/854
	B	US-6,931,532	08-2005	Davis et al.	713/167
	C	US-6,317,742	11-2001	Nagaratnam et al.	707/9
	D	US-5,778,365	07-1998	Nishiyama, Kenji	707/9
	E	US-6,859,671	02-2005	Brown, David W.	700/56
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
*	U	Lennox et al; November 14, 2000; "A Language for User Control of Internet Telephony Services"; November 14, 2000; IETF, Internet Draft; draft-ietf-iptel-cpl-04.txt; Pages 1-60.
*	V	"http://www.iptel.org/info/players/ietf/callprocessing/interfaces/rfc2824.txt"; May 2000; Pages 1-24.
*	W	Danielsen, P.J.; "The promise of a voice-enabled Web"; August 2000; Computer, Volume 33, Issue 8; Pages 104-106.
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

APPENDIX C

Non-Patent Literature entitled "A Language for User Control of Internet Telephony Services" (*Lennox et al.*)

Internet Engineering Task Force
Internet Draft
draft-ietf-iptel-cpl-04.txt
November 14, 2000
Expires: May, 2001

IPTEL WG
Lennox/Schulzrinne
Columbia University

CPL: A Language for User Control of Internet Telephony Services

STATUS OF THIS MEMO

Abstract

The Call Processing Language (CPL) is a language that can be used to describe and control Internet telephony services. It is designed to be implementable on either network servers or user agent servers. It is meant to be simple, extensible, easily edited by graphical clients, and independent of operating system or signalling protocol. It is suitable for running on a server where users may not be allowed to execute arbitrary programs, as it has no variables, loops, or ability to run external programs.

This document is a product of the IP Telephony (IPTEL) working group of the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at iptel@lists.research.bell-labs.com and/or the authors.

1 Introduction

The Call Processing Language (CPL) is a language that can be used to describe and control Internet telephony services. It is not tied to any particular signalling architecture or protocol; it is anticipated that it will be used with both SIP [1] and H.323 [2].

The CPL is powerful enough to describe a large number of services and features, but it is limited in power so that it can run safely in Internet telephony servers. The intention is to make it impossible for users to do anything more complex (and dangerous) than describing Internet telephony services. The language is not Turing-complete, and provides no way to write loops or recursion.

The CPL is also designed to be easily created and edited by graphical tools. It is based on XML [3], so parsing it is easy and many parsers for it are publicly available. The structure of the language maps closely to its behavior, so an editor can understand any valid script, even ones written by hand. The language is also designed so that a server can easily confirm scripts' validity at the time they are delivered to it, rather that discovering them while a call is being processed.

Implementations of the CPL are expected to take place both in Internet telephony servers and in advanced clients; both can usefully process and direct users' calls. This document primarily addresses the usage in servers. A mechanism will be needed to transport scripts between clients and servers; this document does not describe such a mechanism, but related documents will.

The framework and requirements for the CPL architecture are described in RFC 2824, "Call Processing Language Framework and Requirements" [4].

1.1 Conventions of This Document

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [5] and indicate requirement levels for compliant CPL implementations.

> Some paragraphs are indented, like this; they give motivations of design choices, or questions for future discussion in the development of the CPL, and are not essential to the specification of the language.

2 Structure of CPL Scripts

2.1 High-level Structure

A CPL script consists of two types of information: ancillary information about the script, and call processing actions.

A call processing action is a structured tree that describes the operations and decisions a telephony signalling server performs on a call set-up event. There are two types of call processing actions: top-level actions and subactions. Top-level actions are actions that are triggered by signalling events that arrive at the server. Two top-level action names are defined: incoming, the action performed when a call arrives whose destination is the owner of the script; and outgoing, the action performed when a call arrives whose originator is the owner of the script. Subactions are actions which can be called from other actions. The CPL forbids subactions from being called recursively: see Section 9.

Ancillary information is information which is necessary for a server to correctly process a script, but which does not directly describe any operations or decisions. Currently, no ancillary information is defined, but the section is reserved for use by extensions.

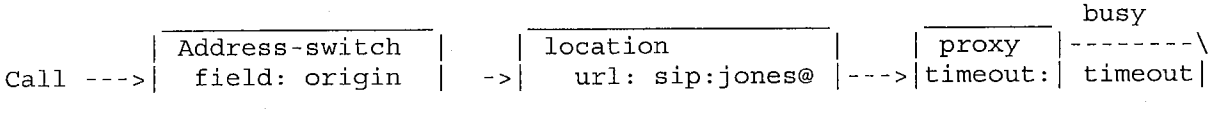
2.2 Abstract Structure of a Call Processing Action

Abstractly, a call processing action is described by a collection of nodes, which describe operations that can be performed or decisions which can be made. A node may have several parameters, which specify the precise behavior of the node; they usually also have outputs, which depend on the result of the decision or action.

For a graphical representation of a CPL action, see Figure 1. Nodes and outputs can be thought of informally as boxes and arrows; the CPL is designed so that actions can be conveniently edited graphically using this representation. Nodes are arranged in a tree, starting at a single root node; outputs of nodes are connected to additional nodes. When an action is run, the action or decision described by the action's top-level node is performed; based on the result of that node, the server follows one of the node's outputs, and the subsequent node it points to is performed; this process continues until a node with no specified outputs is reached. Because the graph is acyclic, this will occur after a bounded and predictable number of nodes are visited.

If an output to a node does not point to another node, it indicates that the CPL server should perform a node- or protocol-specific action. Some nodes have specific default behavior associated with them; for others, the default behavior is implicit in the underlying signalling protocol, or can be configured by the administrator of the

server. For further details on this, see Section 11.



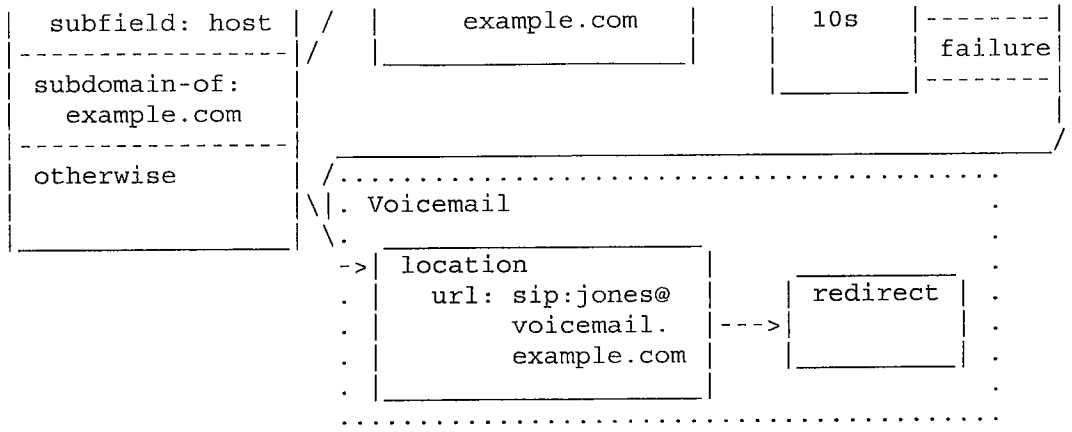


Figure 1: Sample CPL Action: Graphical Version

2.3 Location Model

For flexibility, one piece of information necessary for the function of a CPL is not given as node parameters: the set of locations to which a call is to be directed. Instead, this set of locations is stored as an implicit global variable throughout the execution of a processing action (and its subactions). This allows locations to be retrieved from external sources, filtered, and so forth, without requiring general language support for such operations (which could harm the simplicity and tractability of understanding the language). The specific operations which add, retrieve, or filter location sets are given in Section 6.

For the incoming top-level call processing action, the location set is initialized to the empty set. For the outgoing action, it is initialized to the destination address of the call.

2.4 XML Structure

Syntactically, CPL scripts are represented by XML documents. XML is thoroughly specified by [3], and implementors of this specification should be familiar with that document, but as a brief overview, XML consists of a hierarchical structure of tags; each tag can have a number of attributes. It is visually and structurally very similar to HTML [6], as both languages are simplifications of the earlier and larger standard SGML [7].

See Figure 2 for the XML document corresponding to the graphical representation of the CPL script in Figure 1. Both nodes and outputs in the CPL are represented by XML tags; parameters are represented by XML tag attributes. Typically, node tags contain output tags, and

vice-versa (with a few exceptions: see Sections 6.1, 6.3, 8.1, and 8.2).

The connection between the output of a node and another node is represented by enclosing the tag representing the pointed-to node inside the tag for the outer node's output. Convergence (several outputs pointing to a single node) is represented by subactions, discussed further in Section 9.

The higher-level structure of a CPL script is represented by tags corresponding to each piece of ancillary information, subactions, and top-level actions, in order. This higher-level information is all enclosed in a special tag cpl, the outermost tag of the XML document.

A complete Document Type Declaration for the CPL is provided in Appendix C. The remainder of the main sections of this document describe the semantics of the CPL, while giving its syntax informally. For the formal syntax, please see the appendix.

3 Document Information

This section gives information describing how CPL scripts are identified.

3.1 CPL Document Identifiers for XML

A CPL script list which appears as a top-level XML document is identified with the formal public identifier "-//IETF//DTD RFCxxxx CPL 1.0//EN".

A CPL embedded as a fragment within another XML document is identified with the XML namespace identifier "http://www.rfc-editor.org/rfc/rfcxxxx.txt".


```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <subaction id="voicemail">
    <location url="sip:jones@voicemail.example.com">
      <redirect />
    </location>
  </subaction>

  <incoming>
    <address-switch field="origin" subfield="host">
      <address subdomain-of="example.com">
        <location url="sip:jones@example.com">
          <proxy timeout="10">
```

```
            <busy> <sub ref="voicemail" /> </busy>
            <noanswer> <sub ref="voicemail" /> </noanswer>
            <failure> <sub ref="voicemail" /> </failure>
          </proxy>
        </location>
      </address>
      <otherwise>
        <sub ref="voicemail" />
      </otherwise>
    </address-switch>
  </incoming>
</cpl>
```

Figure 2: Sample CPL Script: XML Version

[Note to RFC editor: please replace "xxxx" above with the number of this RFC.]

Note that the URIs specifying XML namespaces are only globally unique names; they do not have to reference any particular actual object. The URI of a canonical source of this specification meets the requirement of being globally unique, and is also useful to document the format.

3.2 MIME Registration

As an XML type, CPL's MIME registration conforms with "XML Media Types," RFC YYYY [8].

[Note to RFC Editor: please replace "YYYY" in this section, and in bibliography entry [8], with the RFC number assigned to the Internet-Draft draft-murata-xml-09.txt, approved for Proposed Standard.]

MIME media type name: application

MIME subtype name: cpl+xml

Mandatory parameters: none

Optional parameters: charset
As for application/xml in RFC YYYY.

Encoding considerations: As for application/xml in RFC YYYY.

Security considerations: See Section 14, and Section 10 of RFC YYYY.

Interoperability considerations: Different CPL servers may use

incompatible address types. However, all potential interoperability issues should be resolvable at the time a script is uploaded; there should be no interoperability issues which cannot be detected until runtime.

Published specification: This document.

Applications which use this media type: None publicly released at this time, as far as the authors are aware.

Additional information:

Magic number: None

File extension: .cpl or .xml

Macintosh file type code: "TEXT"

Person and e-mail address for further information:
Jonathan Lennox <lennox@cs.columbia.edu>
Henning Schulzrinne <hgs@cs.columbia.edu>

Intended usage: COMMON

Author/Change Controller: The IETF.

4 Script Structure: Overview

As mentioned, a CPL script consists of ancillary information, subactions, and top-level actions. The full syntax of the cpl node is given in Figure 3.

Tag:	cpl	
Parameters:	None	
Sub-tags:	ancillary	See Section 10
	subaction	See Section 9
	outgoing	Top-level actions to take on this user's outgoing calls
	incoming	Top-level actions to take on this user's incoming calls

Figure 3: Syntax of the top-level cpl tag

Call processing actions, both top-level actions and sub-actions, consist of a tree of nodes and outputs. Nodes and outputs are both described by XML tags. There are four categories of CPL nodes: switches , which represent choices a CPL script can make; location modifiers , which add or remove locations from the location set;

signalling operations , which cause signalling events in the underlying protocol; and non-signalling operations , which trigger behavior which does not effect the underlying protocol.

5 Switches

Switches represent choices a CPL script can make, based on either attributes of the original call request or items independent of the call.

All switches are arranged as a list of conditions that can match a variable. Each condition corresponds to a node output; the output points to the next node to execute if the condition was true. The conditions are tried in the order they are presented in the script; the output corresponding to the first node to match is taken.

There are two special switch outputs that apply to every switch type. The output not-present, which MAY occur anywhere in the list of outputs, is true if the variable the switch was to match was not present in the original call setup request. (In this document, this is sometimes described by saying that the information is "absent".) The output otherwise, which MUST be the last output specified if it is present, matches if no other condition matched.

If no condition matches and no otherwise output was present in the script, the default script behavior is taken. See Section 11 for more information on this.

5.1 Address Switches

Address switches allow a CPL script to make decisions based on one of the addresses present in the original call request. They are summarized in Figure 4.

Node:	address-switch	
Outputs:	address	Specific addresses to match
Parameters:	field	origin, destination, or original-destination
	subfield	address-type, user, host, port, tel, or display (also: password and alias-type)
Output:	address	
Parameters:	is	exact match
	contains	substring match (for display only)
	subdomain-of	sub-domain match (for host, tel only)

Figure 4: Syntax of the address-switch node

Address switches have two node parameters: field, and subfield. The

mandatory field parameter allows the script to specify which address is to be considered for the switch: either the call's origin address (field "origin"), its current destination address (field "destination"), or its original destination (field "original-destination"), the destination the call had before any earlier forwarding was invoked. Servers MAY define additional field values.

The optional subfield specifies what part of the address is to be considered. The possible subfield values are: address-type, user, host, port, tel, and display. Additional subfield values MAY be defined for protocol-specific values. (The subfield password is defined for SIP in Section 5.1.1; the subfield alias-type is defined for H.323 in Appendix B.1.) If no subfield is specified, the "entire" address is matched; the precise meaning of this is defined for each underlying signalling protocol. Servers MAY define additional subfield values.

The subfields are defined as follows:

address-type This indicates the type of the underlying address;

i.e., the URI scheme, if the address can be represented by a URI. The types specifically discussed by this document are sip, tel, and h323. The address type is not case-sensitive. It has a value for all defined address types.

user This subfield of the address indicates, for e-mail style addresses, the user part of the address. For telephone number style address, it includes the subscriber number. This subfield is case-sensitive; it may be absent.

host This subfield of the address indicates the Internet host name or IP address corresponding to the address, in host name, IPv4, or IPv6 format. For host names only, subdomain matching is supported with the subdomain-of match operator. It is not case sensitive, and may be absent.

port This subfield indicates the TCP or UDP port number of the address, numerically in decimal format. It is not case sensitive, as it MUST only contain decimal digits. It may be absent; however, for address types with default ports, an absent port matches the default port number.

tel This subfield indicates a telephone subscriber number, if the address contains such a number. It is not case sensitive (the telephone numbers may contain the symbols `A' `B' `C' and `D'), and may be absent. It may be matched using the subdomain-of match operator. Punctuation and separator characters in telephone numbers are discarded.

display This subfield indicates a "display name" or user-visible name corresponding to an address. It is a Unicode string, and is matched using the case-insensitive algorithm

described in Section 5.2. The contains operator may be applied to it. It may be absent.

For any completely unknown subfield, the server MAY reject the script at the time it is submitted with an indication of the problem; if a script with an unknown subfield is executed, the server MUST consider the not-present output to be the valid one.

The address output tag may take exactly one of three possible parameters, indicating the kind of matching allowed.

is An output with this match operator is followed if the subfield being matched in the address-switch exactly matches the argument of the operator. It may be used for any subfield, or for the entire address if no subfield was specified.

subdomain-of This match operator applies only for the subfields host and tel. In the former case, it matches if the hostname being matched is a subdomain of the domain given in the argument of the match operator; thus, subdomain-of="example.com" would match the hostnames "example.com", "research.example.com", and "zaphod.sales.internal.example.com". IP addresses may be given as arguments to this operator; however, they only match exactly. In the case of the tel subfield, the output matches if the telephone number being matched has a prefix that matches the argument of the match operator; subdomain-of="1212555" would match the telephone number "1 212 555 1212."

contains This match operator applies only for the subfield display. The output matches if the display name being matched contains the argument of the match as a substring.

5.1.1 Usage of address-switch with SIP

For SIP, the origin address corresponds to the address in the From header; destination corresponds to the Request-URI; and original-destination corresponds to the To header.

The display subfield of an address is the display-name part of the address, if it is present. Because of SIP's syntax, the destination address field will never have a display subfield.

The address-type subfield of an address is the URI scheme of that address. Other address fields depend on that address-type.

For sip URLs, the user, host, and port subfields correspond to the "user," "host," and "port" elements of the URI syntax. The tel subfield is defined to be the "user" part of the URI, with visual separators stripped,

if and only if the "user=phone" parameter is given to the URI. An additional subfield, password is defined to correspond to the "password" element of the SIP URI, and is case-sensitive. However, use of this field is NOT RECOMMENDED for general security reasons.

For tel URLs, the tel and user subfields are the subscriber name; in the former case, visual separators are stripped. The host and port subfields are both not present.

For h323 URLs, subfields MAY be set according to the scheme described in Appendix B.

For other URI schemes, only the address-type subfield is defined by this specification; servers MAY set other pre-defined subfields, or MAY support additional subfields.

If no subfield is specified for addresses in SIP messages, the string matched is the URI part of the address. For "sip" URLs, all parameters are stripped; for other URLs, the URL is used verbatim.

5.2 String Switches

String switches allow a CPL script to make decisions based on free-form strings present in a call request. They are summarized in Figure 5.

Node:	string-switch	
Outputs:	string	Specific string to match
Parameters:	field	subject, organization, user-agent, language, or display
Output:	string	
Parameters:	is	exact match
	contains	substring match

Figure 5: Syntax of the string-switch node

String switches have one node parameter: field. The mandatory field parameter specifies which string is to be matched.

String switches are dependent on the call signalling protocol being used.

Five fields are defined, listed below. The value of each of these fields, except as specified, is a free-form Unicode string with no other structure defined.

subject The subject of the call.

organization The organization of the originator of the call.

user-agent The name of the program or device with which the call request was made.

language The languages in which the originator of the call wishes to receive responses. This contains a list of RFC

1766 [9] language tags, separated by commas.

Note that matching based on contains is likely to be much more useful than matching based on is, for this field.

display Free-form text associated with the call, intended to be displayed to the recipient, with no other semantics defined by the signalling protocol.

Strings are matched as case-insensitive Unicode strings, in the following manner. First, strings are canonicalized to the "Compatibility Composition" (KC) form, as specified in Unicode Technical Report 15 [10]. Then, strings are compared using locale-insensitive caseless mapping, as specified in Unicode Technical Report 21 [11].

Code to perform the first step, in Java and Perl, is available; see the links from Annex E of UTR 15 [10]. The case-insensitive string comparison in the Java standard class libraries already performs the second step; other Unicode-aware libraries should be similar.

The output tags of string matching are named string, and have a mandatory argument, one of is or contains, indicating whole-string match or substring match, respectively.

5.2.1 Usage of string-switch with SIP

For SIP, the fields subject, organization, and user-agent correspond to the SIP header fields with the same name. These are used verbatim as they appear in the message.

The field language corresponds to the SIP Accept-Language header. It is converted to a list of comma-separated languages as described above.

The field display is not used, and is never present.

5.3 Time Switches

Time switches allow a CPL script to make decisions based on the time

and/or date the script is being executed. They are summarized in Figure 6.

Time switches are independent of the underlying signalling protocol.

Node:	time-switch	
Outputs:	time	Specific time to match
Parameters:	tzid	RFC 2445 Time Zone Identifier
	tzurl	RFC 2445 Time Zone URL
Output:	time	
Parameters:	dtstart	Start of interval (RFC 2445 DATE-TIME)
	dtend	End of interval (RFC 2445 DATE-TIME)
	duration	Length of interval (RFC 2445 DURATION)
	freq	Frequency of recurrence (one of "daily", "weekly", "monthly", or "yearly")
	interval	How often the recurrence repeats
	until	Bound of recurrence (RFC 2445 DATE-TIME)
	byday	List of days of the week
	bymonthday	List of days of the month
	byyearday	List of days of the year
	byweekno	List of weeks of the year
	bymonth	List of months of the year
	wkst	First day of workweek

Figure 6: Syntax of the time-switch node

Time switches are based on a large subset of how recurring intervals of time are specified in the Internet Calendaring and Scheduling Core Object Specification (iCal COS), RFC 2445 [12].

> This allows CPLs to be generated automatically from calendar books. It also allows us to re-use the extensive existing work specifying time intervals.

> The subset was designed with the goal that a time-switch can be evaluated -- an instant can be determined to fall within an interval, or not -- in constant (O(1)) time.

An algorithm to whether an instant falls within a given recurrence is given in Appendix A.

The time-switch tag takes two optional parameters, tzid and tzurl, both of which are defined in RFC 2445 (Sections 4.8.3.1 and 4.8.3.5 respectively). The TZID is the identifying label by which a time zone definition is referenced. If it begins with a forward slash (solidus), it references a to-be-defined global time zone registry;

otherwise it is locally-defined at the server. The TZURL gives a network location from which an up-to-date VTIMEZONE definition for the timezone can be retrieved.

While TZID labels that do not begin with a forward slash are locally defined, it is RECOMMENDED that servers support at least the naming scheme used by Olson Time Zone database [13]. Examples of timezone databases that use the Olson scheme are the zoneinfo files on most Unix-like systems, and the standard Java TimeZone class.

If a script is uploaded with a tzid and tzurl which the CPL server does not recognize or cannot resolve, it SHOULD diagnose and reject this at script upload time. If neither tzid nor tzurl are present, all non-UTC times within this time switch should be interpreted as being "floating" times, i.e. that they are specified in the local timezone of the CPL server.

> Because of daylight-savings-time changes over the course of a year, it is necessary to specify time switches in a given timezone. UTC offsets are not sufficient, or a time-of-day routing rule which held between 9 am and 5 pm in the eastern United States would start holding between 8 am and 4 pm at the end of October.

Authors of CPL servers should be careful to handle correctly the intervals when local time is discontinuous, at the beginning or end of daylight-savings time. Note especially that some times may occur more than once when clocks are set back. The algorithm in Appendix A is believed to handle this correctly.

Time nodes specify a list of periods during which their output should be taken. They have two required parameters: dtstart, which specifies the beginning of the first period of the list, and exactly one of dtend or duration, which specify the ending time or the duration of the period, respectively. The dtstart and dtend parameters are formatted as iCal COS DATE-TIME values, as specified in Section 4.3.5 of RFC 2445 [12]. Because time zones are specified in the top-level time-switch tag, only forms 1 or 2 (floating or UTC times) can be used. The duration parameter is given as an iCal COS DURATION parameter, as specified in section 4.3.6 of RFC 2445. Both the DATE-TIME and the DURATION syntaxes are subsets of the corresponding syntaxes from ISO 8601 [14].

For a recurring interval, the duration parameter MUST be less than twenty-four hours. For non-recurring intervals, durations of any length are permitted.

If no other parameters are specified, a time node indicates only a single period of time. More complicated sets periods intervals are constructed as recurrences. A recurrence is specified by including the freq parameter, which indicates the type of recurrence rule. No parameters other than dtstart, dtend, and duration SHOULD be specified unless freq is present.

The freq parameter takes one of the following values: daily, to specify repeating periods based on an interval of a day or more; weekly, to specify repeating periods based on an interval of a week or more; monthly, to specify repeating periods based on an interval of a month or more; and yearly, to specify repeating periods based on an interval of a year or more. These values are not case-sensitive.

> The values secondly, minutely, and hourly are present in iCal, but were removed from CPL.

The interval parameter contains a positive integer representing how often the recurrence rule repeats. The default value is "1", meaning every day for a daily rule, every week for a weekly rule, every month for a monthly rule and every year for a yearly rule.

The until parameter defines an iCal COS DATE or DATE-TIME value which bounds the recurrence rule in an inclusive manner. If the value specified by until is synchronized with the specified recurrence, this date or date-time becomes the last instance of the recurrence. If specified as a date-time value, then it MUST be specified in an UTC time format. If not present, the recurrence is considered to repeat forever.

> iCal also defines a count parameter, which allows an alternate method of specifying a bound to a recurrence. This bound has been removed from CPL. Translating from full iCal recurrences to CPL recurrences requires that the count parameter be converted to an until parameter, which can be done by enumerating the recurrence and determining its final date.

The byday parameter specifies a comma-separated list of days of the week. MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; SA indicates Saturday; SU indicates Sunday. These values are not case-sensitive.

Each byday value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific day within the monthly or yearly recurrence. For example,

within a monthly rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a monthly rule, MO represents all Mondays within the month.

The bymonthday parameter specifies a comma-separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month.

The byyearday parameter specifies a comma-separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st).

The byweekno parameter specifies a comma-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. This corresponds to weeks according to week numbering as defined in ISO 8601 [14]. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see wkst). Week number one of the calendar year is the first week which contains at least four (4) days in that calendar year. This parameter is only valid for yearly rules. For example, 3 represents the third week of the year.

> Note: Assuming a Monday week start, week 53 can only occur when Thursday is January 1 or if it is a leap year and Wednesday is January 1.

The bymonth parameter specifies a comma-separated list of months of the year. Valid values are 1 to 12.

The wkst parameter specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA and SU. This is significant when a weekly recurrence has an interval greater than 1, and a byday parameter is specified. This is also significant in a yearly recurrence when a byweekno parameter is specified. The default value is MO, following ISO 8601 [14].

> iCal also includes the Byxxx parameters bysecond, byminute, byhour, and bysetpos, which have been removed from CPL.

If byxxx parameter values are found which are beyond the available scope (ie, bymonthday="30" in February), they are simply ignored.

Byxxx parameters modify the recurrence in some manner. Byxxx rule

parts for a period of time which is the same or greater than the frequency generally reduce or limit the number of occurrences of the recurrence generated. For example, freq="daily" bymonth="1" reduces the number of recurrence instances from all days (if the bymonth

parameter is not present) to all days in January. Byxxx parameters for a period of time less than the frequency generally increase or expand the number of occurrences of the recurrence. For example, freq="yearly" bymonth="1,2" increases the number of days within the yearly recurrence set from 1 (if bymonth parameter is not present) to 2.

If multiple Byxxx parameters are specified, then after evaluating the specified freq and interval parameters, the Byxxx parameters are applied to the current set of evaluated occurrences in the following order: bymonth, byweekno, byyearday, bymonthday, and byday; then until is evaluated.

Here is an example of evaluating multiple Byxxx parameters.

```
<time dtstart="19970105T083000" duration="PT10M"
      freq="yearly" interval="2" bymonth="1" byday="SU">
```

First, the interval="2" would be applied to freq="YEARLY" to arrive at "every other year." Then, bymonth="1" would be applied to arrive at "every January, every other year." Then, byday="SU" would be applied to arrive at "every Sunday in January, every other year." Then the time of day is derived from dtstart to end up in "every Sunday in January from 8:30:00 AM to 8:40:00 AM, every other year." Similarly, if the byday, bymonthday or bymonth parameter were missing, the appropriate day or month would have been retrieved from the dtstart parameter.

The iCal COS RDATE, EXRULE and EXDATE recurrence rules are not specifically mapped to components of the time-switch node. Equivalent functionality to the exception rules can be attained by using the ordering of switch rules to exclude times using earlier rules; equivalent functionality to the additional-date RDATE rules can be attained by using sub nodes (see Section 9) to link multiple outputs to the same subsequent node.

The not-present output is never true for a time switch. However, it MAY be included, to allow switch processing to be more regular.

5.3.1 Motivations for the iCal Subset

(This sub-sub-section is non-normative.)

The syntax of the CPL time-switch was based on that of the iCal COS RRULE, but as mentioned above, certain features were omitted and restrictions were added. Specifically:

1. All recurrence intervals and rules describing periods less than a day were removed. These were the frequencies

secondly, minutely, and hourly, and the Byxxx rules bysecond, byminute, and byhour.

2. The count and bysetpos parameters were removed.

3. Durations were constrained to less than 24 hours for recurring intervals.

These restrictions were added so that time switches could be resolved efficiently, in O(1) time. This restriction means that it must be possible to resolve a time switch without having to enumerate all its recurrences from dtstart to the present interval. As far as we have been able to determine, it is not possible to test whether the count and bysetpos parameters are satisfied without performing such an enumeration.

Constant running time of time switches also requires that a candidate starting time for a recurrence can be established quickly and uniquely, to check whether it satisfies the other restrictions. This requires that a recurrence's duration not be longer than its repetition interval, so that a given instant cannot fall within several consecutive repetitions of the recurrence. We guaranteed this by eliminating durations longer than 24 hours, and repetitions shorter than that period. The one-day point seemed to be the most generally useful place to place this division, as some investigation showed that many common calendaring applications do not support durations longer than a day, none that we found supported repetitions shorter than a day. Eliminating sub-day repetitions also greatly simplifies the handling of daylight-savings transitions.

The algorithm given in Appendix A runs in constant time, and motivated the development of this iCal subset.

5.4 Priority Switches

Priority switches allow a CPL script to make decisions based on the priority specified for the original call. They are summarized in Figure 7. They are dependent on the underlying signalling protocol.

Node:	priority-switch	
Outputs:	priority	Specific priority to match
Parameters:	None	
Output:	priority	
Parameters:	less	Match if priority is less than specified
	greater	Match if priority is greater than specified
	equal	Match if priority is equal to specified

Figure 7: Syntax of the priority-switch node

Priority switches take no parameters.

The priority tags take one of the three parameters greater, less, and equal. The values of these tags are one of the following priorities: in decreasing order, emergency, urgent, normal, and non-urgent. These values are matched in a case-insensitive manner. Outputs with the less parameter are taken if the priority of the call is less than the priority given in the argument; and so forth.

If no priority header is specified in a message, the priority is considered to be normal. If an unknown priority is specified in the call, it is considered to be equivalent to normal for the purposes of greater and less comparisons, but it is compared literally for equal comparisons.

Since every message has a priority, the not-present output is never true for a priority switch. However, it MAY be included, to allow switch processing to be more regular.

5.4.1 Usage of priority-switch with SIP

The priority of a SIP message corresponds to the Priority header in the initial INVITE message.

6 Location Modifiers

The abstract location model of the CPL is described in Section 2.3. The behavior of several of the signalling operations (defined in Section 7) is dependent on the current location set specified. Location nodes add or remove locations from the location set.

There are three types of location nodes defined. Explicit locations add literally-specified locations to the current location set; location lookups obtain locations from some outside source; and

location filters remove locations from the set, based on some specified criteria.

6.1 Explicit Location

Explicit location nodes specify a location literally. Their syntax is described in Figure 8.

Explicit location nodes are dependent on the underlying signalling protocol.

Node:	location
Outputs:	None (next node follows directly)
Next node:	Any node
Parameters:	url URL of address to add to locati

```
                priority                Priority of this location (0.0-
                clear                   Whether to clear the location s
the new value
```

Figure 8: Syntax of the location node

Explicit location nodes have three node parameters. The mandatory url parameter's value is the URL of the address to add to the location set. Only one address may be specified per location node; multiple locations may be specified by cascading these nodes.

The optional priority parameter specifies a priority for the location. Its value is a floating-point number between 0.0 and 1.0. If it is not specified, the server SHOULD assume a default priority of 1.0. The optional clear parameter specifies whether the location set should be cleared before adding the new location to it. Its value can be "yes" or "no", with "no" as the default.

Basic location nodes have only one possible result, since there is no way that they can fail. (If a basic location node specifies a location which isn't supported by the underlying signalling protocol, the script server SHOULD detect this and report it to the user at the time the script is submitted.) Therefore, their XML representations do not have explicit output tags; the <location> tag directly contains another node.

6.1.1 Usage of location with SIP

All SIP locations are represented as URLs, so the locations specified

in location tags are interpreted directly.

6.2 Location Lookup

Locations can also be specified up through external means, through the use of location lookups. The syntax of these tags is given in Figure 9.

Location lookup is dependent on the underlying signalling protocol.

```
          Node:  lookup
       Outputs:  success    Next node if lookup was successful
                 notfound   Next node if lookup found no addresses
                 failure    Next node if lookup failed
    Parameters:  source     Source of the lookup
                 timeout    Time to try before giving up on the lookup
                 use        Caller preferences fields to use
                 ignore     Caller preferences fields to ignore
                 clear      Whether to clear the location set before adding
```

the new values

```
       Output:  success
   Parameters:  none

       Output:  notfound
   Parameters:  none

       Output:  failure
   Parameters:  none
```

Figure 9: Syntax of the lookup node

Location lookup nodes have one mandatory parameter, and four optional parameters. The mandatory parameter is source, the source of the lookup. This can either be a URI, or a non-URI value. If the value of source is a URI, it indicates a location which the CPL server can query to obtain an object with the text/uri-list media type (see the IANA registration of this type, which also appears in RFC 2483 [15]). The query is performed verbatim, with no additional information (such as URI parameters) added. The server adds the locations contained in this object to the location set.

CPL servers MAY refuse to allow URI-based sources for location queries for some or all URI schemes. In this case, they SHOULD reject

the script at script upload time.

> There has been discussion of having CPL servers add URI parameters to the location request, so that (for instance) CGI scripts could be used to resolve them. However, the consensus was that this should be a CPL extension, not a part of the base specification.

Non-URL sources indicate a source not specified by a URL which the server can query for addresses to add to the location set. The only non-URL source currently defined is registration, which specifies all the locations currently registered with the server.

The lookup node also has four optional parameters. The timeout parameter specifies the time, in seconds, the script is willing to wait for the lookup to be performed. If this is not specified, its default value is 30. The clear parameter specifies whether the location set should be cleared before the new locations are added.

The other two optional parameters affect the interworking of the CPL script with caller preferences and caller capabilities. By default, a CPL server SHOULD invoke the appropriate caller preferences filtering of the underlying signalling protocol, if the corresponding

information is available. The two parameters use and ignore allow the script to modify how the script applies caller preferences filtering. The specific meaning of the values of these parameters is signalling-protocol dependent; see Section 6.2.1 for SIP and Appendix B.5 for H.323.

Lookup has three outputs: success, notfound, and failure. Notfound is taken if the lookup process succeeded but did not find any locations; failure is taken if the lookup failed for some reason, including that specified timeout was exceeded. If a given output is not present, script execution terminates and the default behavior is performed.

Clients SHOULD specify the three outputs success, notfound, and failure in that order, so their script complies with the DTD given in Appendix C, but servers MAY accept them in any order.

6.2.1 Usage of lookup with SIP

Caller preferences for SIP are defined in "SIP Caller Preferences and Callee Capabilities" [16]. By default, a CPL server SHOULD honor any Accept-Contact and Reject-Contact headers of the original call request, as specified in that document. The two parameters use and ignore allow the script to modify the data input to the caller preferences algorithm. These parameters both take as their arguments

comma-separated lists of caller preferences parameters. If use is given, the server applies the caller preferences resolution algorithm only to those preference parameters given in the use parameter, and ignores all others; if the ignore parameter is given, the server ignores the specified parameters, and uses all the others. Only one of use and ignore can be specified.

The addr-spec part of the caller preferences is always applied, and the script cannot modify it.

If a SIP server does not support caller preferences and callee capabilities, if the call request does not contain any preferences, or if the callee's registrations do not contain any capabilities, the use and ignore parameters are ignored.

6.3 Location Removal

A CPL script can also remove locations from the location set, through the use of the remove-location node. The syntax of this node is defined in Figure 10.

The meaning of this node is dependent on the underlying signalling protocol.

```
      Node:  remove-location
   Outputs:  None (next node follows directly)
 Next node:  Any node
```

```
Parameters:  location        Location to remove
             param           Caller preference parameters to a
             value           Value of caller preference parame
```

Figure 10: Syntax of the remove-location node

A remove-location node removes locations from the location set. It is primarily useful following a lookup node. An example of this is given in Section 13.8.

The remove-location node has three optional parameters. The parameter location gives the URL (or a signalling-protocol-dependent URL pattern) of location or locations to be removed from the set. If this parameter is not given, all locations, subject to the constraints of the other parameters, are removed from the set.

If param and value are present, their values are comma-separated

lists of caller preferences parameters and corresponding values, respectively. The nth entry in the param list matches the nth entry in the value list. There MUST be the same number of parameters as values specified. The meaning of these parameters is signalling-protocol dependent.

The remove-location node has no explicit output tags. In the XML syntax, the XML remove-location tag directly encloses the next node's tag.

6.3.1 Usage of remove-location with SIP

For SIP-based CPL servers, the remove-location node has the same effect on the location set as a Reject-Contact header in caller preferences [16]. The value of the location parameter is treated as though it were the addr-spec field of a Reject-Contact header; thus, an absent header is equivalent to an addr-spec of "*" in that specification. The param and value parameters are treated as though they appeared in the params field of a Reject-Location header, as ";param=value" for each one.

If the CPL server does not support caller preferences and callee capabilities, or if the callee did not supply any preferences, the param and value parameters are ignored.

7 Signalling Operations

Signalling operation nodes cause signalling events in the underlying signalling protocol. Three signalling operations are defined: "proxy," "redirect," and "reject."

7.1 Proxy

Proxy causes the triggering call to be forwarded on to the currently specified set of locations. The syntax of the proxy node is given in Figure 11.

The specific signalling events invoked by the proxy node are signalling-protocol-dependent, though the general concept should apply to any signalling protocol.

After a proxy operation has completed, the CPL server chooses the "best" response to the call attempt, as defined by the signalling protocol or the server's administrative configuration rules.

If the call attempt was successful, CPL execution terminates and the server proceeds to its default behavior (normally, to allow the call


```
      Node:  proxy
   Outputs:  busy         Next node if call attempt returned "busy"
             noanswer     Next node if call attempt was not answered before timeo
             redirection  Next node if call attempt was redirected
             failure      Next node if call attempt failed
             default      Default next node for unspecified outputs
Parameters:  timeout      Time to try before giving up on the call attempt
             recurse      Whether to recursively look up redirections
             ordering     What order to try the location set in.

    Output:  busy
Parameters:  none

    Output:  noanswer
Parameters:  none

    Output:  redirection
Parameters:  none

    Output:  failure
Parameters:  none

    Output:  default
Parameters:  none
```

Figure 11: Syntax of the proxy node

to be set up). Otherwise, the next node corresponding to one of the proxy node's outputs is taken. The busy output is followed if the call was busy; noanswer is followed if the call was not answered before the timeout parameter expired; redirection is followed if the call was redirected; and failure is followed if the call setup failed for any other reason.

If one of the conditions above is true, but the corresponding output was not specified, the default output of the proxy node is followed instead. If there is also no default node specified, CPL execution terminates and the server returns to its default behavior (normally, to forward the best response upstream to the originator).

> Note: CPL extensions to allow in-call or end-of-call operations will require an additional output, such as success, to be added.

If no locations were present in the set, or if the only locations in the set were locations to which the server cannot proxy a call (for example, "http" URLs), the failure output is taken.

Proxy has three optional parameters. The timeout parameter specifies the time, in seconds, to wait for the call to be completed or rejected; after this time has elapsed, the call attempt is terminated and the noanswer branch is taken. If this parameter is not specified, the default value is 20 seconds if the proxy node has a noanswer or default output specified; otherwise the server SHOULD allow the call to ring for a reasonably long period of time (to the maximum extent that server policy allows).

The second optional parameter is recurse, which can take two values, yes or no. This specifies whether the server should automatically attempt to place further call attempts to telephony addresses in redirection responses that were returned from the initial server. Note that if the value of recurse is yes, the redirection output to the script is never taken. In this case this output SHOULD NOT be present. The default value of this parameter is yes.

The third optional parameter is ordering. This can have three possible values: parallel, sequential, and first-only. This parameter specifies in what order the locations of the location set should be tried. Parallel asks that they all be tried simultaneously; sequential asks that the one with the highest priority be tried first, the one with the next-highest priority second, and so forth, until one succeeds or the set is exhausted. First-only instructs the server to try only the highest-priority address in the set, and then follow one of the outputs. The priority of locations in a set is determined by server policy, though CPL servers SHOULD honor the priority parameter of the location tag. The default value of this parameter is parallel.

Once a proxy operation completes, if control is passed on to other nodes, all locations which have been used are cleared from the location set. That is, the location set is emptied of proxyable locations if the ordering was parallel or sequential; the highest-priority item in the set is removed from the set if ordering was first-only. (In all cases, non-proxyable locations such as "http"

URIs remain.) In the case of a redirection output, the new addresses to which the call was redirected are then added to the location set.

7.1.1 Usage of proxy with SIP

For SIP, the best response to a proxy node is determined by the algorithm of the SIP specification. The node's outputs correspond to the following events:

busy A 486 or 600 response was the best response received to the call request.

redirection A 3xx response was the best response received to the call request.

failure Any other 4xx, 5xx, or 6xx response was the best response received to the call request.

no-answer No final response was received to the call request before the timeout expired.

SIP servers SHOULD honor the q parameter of SIP registrations and the output of the caller preferences lookup algorithm when determining location priority.

7.2 Redirect

Redirect causes the server to direct the calling party to attempt to place its call to the currently specified set of locations. The syntax of this node is specified in Figure 12.

The specific behavior the redirect node invokes is dependent on the underlying signalling protocol involved, though its semantics are generally applicable.

Node:	redirect	
Outputs:	None (no node may follow)	
Next node:	None	
Parameters:	permanent	Whether the redirection should be considered permanent

Figure 12: Syntax of the redirect node

Redirect immediately terminates execution of the CPL script, so this node has no outputs and no next node. It has one parameter, permanent, which specifies whether the result returned should indicate that this is a permanent redirection. The value of this parameter is either "yes" or "no" and its default value is "no."

7.2.1 Usage of redirect with SIP

The SIP server SHOULD send a 3xx class response to a call request upon executing a redirect tag. If permanent was yes, the server

SHOULD send the response "301 Moved permanently"; otherwise it SHOULD send "302 Moved temporarily".

7.3 Reject

Reject nodes cause the server to reject the call attempt. Their syntax is given in Figure 13. The specific behavior they invoke is dependent on the underlying signalling protocol involved, though their semantics are generally applicable.

Node:	reject	
Outputs:	None (no node may follow)	
Next node:	None	
Parameters:	status	Status code to return
	reason	Reason phrase to return

Figure 13: Syntax of the reject node

This immediately terminates execution of the CPL script, so this node has no outputs and no next node.

This node has two arguments: status and reason. The status argument is required, and can take one of the values busy, notfound, reject, and error, or a signalling-protocol-defined status.

The reason argument optionally allows the script to specify a reason for the rejection.

7.3.1 Usage of reject with SIP

Servers which implement SIP SHOULD also allow the status field to be a numeric argument corresponding to a SIP status in the 4xx, 5xx, or 6xx range.

They SHOULD send the "reason" parameter in the SIP reason phrase.

A suggested mapping of the named statuses is as follows. Servers MAY use a different mapping, though similar semantics SHOULD be preserved.

busy: 486 Busy Here

notfound: 404 Not Found

reject: 603 Decline

error: 500 Internal Server Error

8 Non-signalling Operations

In addition to the signalling operations , the CPL defines several operations which do not affect and are not dependent on the telephony signalling protocol.

8.1 Mail

The mail node causes the server to notify a user of the status of the CPL script through electronic mail. Its syntax is given in Figure 14.

Node:	mail	
Outputs:	None (next node follows directly)	
Next node:	Any node	
Parameters:	url	Mailto url to which the mail shou

Figure 14: Syntax of the mail node

The mail node takes one argument: a mailto URL giving the address, and any additional desired parameters, of the mail to be sent. The server sends the message containing the content to the given url; it SHOULD also include other status information about the original call request and the CPL script at the time of the notification.

Using a full mailto URL rather than just an e-mail address allows additional e-mail headers to be specified, such as <mail url="mailto:jones@example.com?subject=lookup%20failed" />.

Mail nodes have only one possible result, since failure of e-mail delivery cannot reliably be known in real-time. Therefore, its XML representation does not have output tags: the <mail> tag directly contains another node tag.

Note that the syntax of XML requires that ampersand characters, "&", which are used as parameter separators in mailto URLs, be quoted as "&" inside parameter values (see Section C.12 of [3]).

8.1.1 Suggested Content of Mailed Information

This section presents suggested guidelines for the mail sent as a result of the mail node, for requests triggered by SIP. The message mailed (triggered by any protocol) SHOULD contain all this information, but servers MAY elect to use a different format.

1. If the mailto URI did not specify a subject header, the subject of the e-mail is "[CPL]" followed by the subject header of the SIP request. If the URI specified a subject header, it is used instead.

2. The From field of the e-mail is set to a CPL server configured address, overriding any From field in the mailto URI.

3. Any Reply-To header in the URI is honored. If none is given, then an e-mail-ized version of the origin field of the request is used, if possible (e.g., a SIP From header with a sip: URI would be converted to an e-mail address by stripping the URI scheme).

4. If the mailto URI specifies a body, it is used. If none was specified, the body SHOULD contain at least the identity of the caller (both the caller's display name and address), the date and time of day, the call subject, and if available, the call priority.

The server SHOULD honor the user's requested languages, and send the mail notification using an appropriate language and character set.

8.2 Log

The Log node causes the server to log information about the call to non-volatile storage. Its syntax is specified in Figure 15.

Node:	log	
Outputs:	None (next node follows directly)	
Next node:	Any node	
Parameters:	name	Name of the log file to use
	comment	Comment to be placed in log file

Figure 15: Syntax of the log node

Log takes two arguments, both optional: name, which specifies the name of the log, and comment, which gives a comment about the

information being logged. Servers SHOULD also include other information in the log, such as the time of the logged event, information that triggered the call to be logged, and so forth. Logs are specific to the owner of the script which logged the event. If the name parameter is not given, the event is logged to a standard, server-defined log file for the script owner. This specification does not define how users may retrieve their logs from the server.

The name of a log is a logical name only, and does not necessarily correspond to any physical file on the server. The interpretation of the log file name is server defined, as is a mechanism to access these logs. The CPL server SHOULD NOT directly map log names uninterpreted onto local file names, for security reasons, lest a security-critical file be overwritten.

A correctly operating CPL server SHOULD NOT ever allow the log event to fail. As such, log nodes can have only one possible result, and their XML representation does not have explicit output tags. A CPL <log> tag directly contains another node tag.

9 Subactions

XML syntax defines a tree. To allow more general call flow diagrams, and to allow script re-use and modularity, we define subactions.

Two tags are defined for subactions: subaction definitions and subaction references. Their syntax is given in Figure 16.

Tag:	subaction	
Subtags:	Any node	
Parameters:	id	Name of this subaction
Pseudo-node:	sub	
Outputs:	None in XML tree	
Parameters:	ref	Name of subaction to execute

Figure 16: Syntax of subactions and sub pseudo-nodes

Subactions are defined through subaction tags. These tags are placed in the CPL after any ancillary information (see Section 10) but before any top-level tags. They take one argument: id, a token indicating a script-chosen name for the subaction.

Subactions are called from sub tags. The sub tag is a "pseudo-node":

it can be used anyplace in a CPL action that a true node could be used. It takes one parameter, ref, the name of the subaction to be called. The sub tag contains no outputs of its own; control instead passes to the subaction.

References to subactions MUST refer to subactions defined before the current action. A sub tag MUST NOT refer to the action which it appears in, or to any action defined later in the CPL script. Top-level actions cannot be called from sub tags, or through any other means. Script servers MUST verify at the time the script is submitted that no sub node refers to any subaction which is not its proper predecessor.

> Allowing only back-references of subs forbids any sort of recursion. Recursion would introduce the possibility of non-terminating or non-decidable CPL scripts, a possibility our requirements specifically excluded.

Every sub MUST refer to a subaction ID defined within the same CPL script. No external links are permitted.

Subaction IDs are case sensitive.

> If any subsequent version or extension defines external linkages, it should probably use a different tag, perhaps XLink [17]. Ensuring termination in the presence of external links is a difficult problem.

10 Ancillary Information

No ancillary information is defined in the base CPL specification. If ancillary information, not part of any operation, is found to be necessary for a CPL extension, it SHOULD be placed within this tag.

The (trivial) definition of the ancillary information tag is given in Figure 17.

> It may be useful to include timezone definitions inside CPL scripts directly, rather than referencing them externally with tzid and tzurl parameters. If it is, an extension could be defined to include them here.

11 Default Behavior


```
          Tag:  ancillary
   Parameters:  None
      Subtags:  None
```

Figure 17: Syntax of the ancillary tag

When a CPL node reaches an unspecified output, either because the output tag is not present, or because the tag is present but does not contain a node, the CPL server's behavior is dependent on the current state of script execution. This section gives the operations that should be taken in each case.

no location modifications or signalling operations performed, location set empty: Look up the user's location through whatever mechanism the server would use if no CPL script were in effect. Proxy, redirect, or send a rejection message, using whatever policy the server would use in the absence of a CPL script.

no location modifications or signalling operations performed, location set non-empty: (This can only happen for outgoing calls.) Proxy the call to the addresses in the location set.

location modifications performed, no signalling operations: Proxy or redirect the call, whichever is the server's standard policy, to the addresses in the current location set. If the location set is empty, return notfound rejection.

noanswer output of proxy, no timeout given: (This is a special case.) If the noanswer output of a proxy node is unspecified, and no timeout parameter was given to the proxy node, the call should be allowed to ring for the maximum length of time allowed by the server (or the request, if the request specified a timeout).

proxy operation previously taken: Return whatever the "best" response is of all accumulated responses to the call to this point, according to the rules of the underlying signalling protocol.

12 CPL Extensions

Servers MAY support additional CPL features beyond those listed in this document. Some of the extensions which have been suggested are a means of querying how a call has been authenticated; richer control over H.323 addressing; end-system or administrator-specific features; regular-expression matching for strings and addresses; mid-call or end-of-call controls; and the parts of iCal COS recurrence rules omitted from time switches.

CPL extensions are indicated by XML namespaces [18]. Every extension MUST have an appropriate XML namespace assigned to it. All XML tags and attributes that are part of the extension MUST be appropriately qualified so as to place them within that namespace.

Tags or attributes in a CPL script which are in the global namespace (i.e., not associated with any namespace) are equivalent to tags and attributes in the CPL namespace "http://www.rfc-editor.org/rfc/rfcxxxx.txt".

A CPL server MUST reject any script which contains a reference to a namespace which it does not understand. It MUST reject any script which contains an extension tag or attribute which is not qualified to be in an appropriate namespace.

A CPL script SHOULD NOT specify any namespaces it does not use. For compatibility with non-namespace-aware parsers, a CPL script SHOULD NOT specify the base CPL namespace for a script which does not use any extensions.

A syntax such as

```
<extension-switch>
  <extension has="http://www.example.com/foo">
     [extended things]
  </extension>
  <otherwise>
     [non-extended things]
  </otherwise>
</extension-switch>
```

was suggested as an alternate way of handling extensions. This would allow scripts to be uploaded to a server without requiring a script author to somehow determine which extensions a server supports. However, experience developing other languages, notably Sieve [19], was that this added excessive complexity to languages. The extension-switch tag could, of course, itself be defined in a CPL extension.

It is unfortunately true that XML DTDs, such as the CPL DTD given in Appendix C, are not powerful enough to encompass namespaces, since the base XML specification (which defines DTDs) predates the XML namespace specification. XML schemas [20] are a work in progress to define a namespace-aware method for validating XML documents, as well as improving upon DTDs' expressive power in many other ways.

13 Examples

13.1 Example: Call Redirect Unconditional

The script in Figure 18 is a simple script which redirects all calls to a single fixed location.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <location url="sip:smith@phone.example.com">
     <redirect />
    </location>
  </incoming>
</cpl>
```

Figure 18: Example Script: Call Redirect Unconditional

13.2 Example: Call Forward Busy/No Answer

The script in Figure 19 illustrates some more complex behavior. We see an initial proxy attempt to one address, with further operations if that fails. We also see how several outputs take the same action subtree, through the use of subactions.

13.3 Example: Call Forward: Redirect and Default

The script in Figure 20 illustrates further proxy behavior. The server initially tries to proxy to a single address. If this attempt is redirected, a new redirection is generated using the locations returned. In all other failure cases for the proxy node, a default operation -- forwarding to voicemail -- is performed.


```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <subaction id="voicemail">
    <location url="sip:jones@voicemail.example.com" >
      <proxy />
    </location>
  </subaction>

  <incoming>
    <location url="sip:jones@jonespc.example.com">
       <proxy timeout="8">
         <busy>
           <sub ref="voicemail" />
         </busy>
         <noanswer>
           <sub ref="voicemail" />
         </noanswer>
```

```
        </proxy>
      </location>
    </incoming>
  </cpl>
```

Figure 19: Example Script: Call Forward Busy/No Answer

13.4 Example: Call Screening

The script in Figure 21 illustrates address switches and call rejection, in the form of a call screening script. Note also that because the address-switch lacks an otherwise clause, if the initial pattern did not match, the script does not define any operations. The server therefore proceeds with its default behavior, which would presumably be to contact the user.

13.5 Example: Priority and Language Routing

The script in Figure 22 illustrates service selection based on a call's priority value and language settings. If the call request had a priority of "urgent" or higher, the default script behavior is performed. Otherwise, the language string field is checked for the string "es" (Spanish). If it is present, the call is proxied to a Spanish-speaking operator; other calls are proxied to an English-


```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <subaction id="voicemail">
  </subaction>

  <incoming>
    <location url="sip:jones@jonespc.example.com">
       <proxy>
         <redirection>
           <redirect />
         </redirection>
         <default>
           <location url="sip:jones@voicemail.example.com" >
              <proxy />
           </location>
         </default>
       </proxy>
    </location>
  </incoming>
</cpl>
```

Figure 20: Example Script: Call Forward: Redirect and Default

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <address-switch field="origin" subfield="user">
      <address is="anonymous">
         <reject status="reject"
                 reason="I don't accept anonymous calls" />
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Figure 21: Example Script: Call Screening

speaking operator.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <priority-switch>
      <priority greater="urgent" />
      <otherwise>
        <string-switch field="language">
          <string contains="es">
            <location url="sip:spanish@operator.example.com">
              <proxy />
            </location>
          </string>
          <otherwise>
            <location url="sip:english@operator.example.com">
              <proxy />
            </location>
          </otherwise>
        </string-switch>
      </otherwise>
    </priority-switch>
  </incoming>
</cpl>
```

Figure 22: Example Script: Priority and Language Routing

13.6 Example: Outgoing Call Screening

The script in Figure 23 illustrates a script filtering outgoing calls, in the form of a script which prevent 1-900 (premium) calls from being placed. This script also illustrates subdomain matching.

13.7 Example: Time-of-day Routing

Figure 24 illustrates time-based conditions and timezones.

13.8 Example: Location Filtering

Figure 25 illustrates filtering operations on the location set. In


```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <outgoing>
    <address-switch field="original-destination" subfield="tel">
      <address subdomain-of="1900">
        <reject status="reject"
                reason="Not allowed to make 1-900 calls." />
      </address>
    </address-switch>
  </outgoing>
</cpl>
```

Figure 23: Example Script: Outgoing Call Screening

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <time-switch tzid="America/New_York"
        tzurl="http://zones.example.com/tz/America/New_York">
      <time dtstart="20000703T090000" duration="PT8H"
            freq="weekly" byday="MO,TU,WE,TH,FR">
        <lookup source="registration">
          <success>
            <proxy />
```

```
            </success>
          </lookup>
        </time>
        <otherwise>
          <location url="sip:jones@voicemail.example.com">
            <proxy />
          </location>
        </otherwise>
      </time-switch>
    </incoming>
  </cpl>
```

Figure 24: Example Script: Time-of-day Routing

this example, we assume that version 0.9beta2 of the "Inadequate Software SIP User Agent" mis-implements some features, and so we must work around its problems. We assume, first, that the value of its

particular mobile device we may have registered, so we remove that location from the location set. Once these two operations have been completed, call setup is allowed to proceed normally.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <string-switch field="user-agent">
      <string is="Inadequate Software SIP User Agent/0.9beta2">
        <lookup source="registration" ignore="feature">
          <success>
            <remove-location location="sip:me@mobile.provider.net">
              <proxy />
            </remove-location>
          </success>
        </lookup>
      </string>
    </string-switch>
  </incoming>
</cpl>
```

Figure 25: Example Script: Location Filtering

13.9 Example: Non-signalling Operations

Figure 26 illustrates non-signalling operations; in particular, alerting a user by electronic mail if the lookup server failed. The primary motivation for having the mail node is to allow this sort of out-of-band notification of error conditions, as the user might otherwise be unaware of any problem.

13.10 Example: Hypothetical Extensions

The example in Figure 27 shows a hypothetical extension which implements distinctive ringing. The XML namespace "http://www.example.com/distinctive-ring" specifies a new node named ring.

The example in Figure 28 implements a hypothetical new attribute for address switches, to allow regular-expression matches. It defines a


```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <lookup source="http://www.example.com/cgi-bin/locate.cgi?user=jones"
            timeout="8">
      <success>
        <proxy />
      </success>
      <failure>
        <mail url="mailto:jones@example.com?subject=lookup%20failed" />
      </failure>
    </lookup>
  </incoming>
</cpl>
```

Figure 26: Example Script: Non-signalling Operations

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl xmlns="http://www.ietf.org/internet-drafts/draft-ietf-iptel-cpl-03.txt"
     xmlns:dr="http://www.example.com/distinctive-ring">
  <incoming>
    <address-switch field="origin">
      <address is="sip:boss@example.com">
         <dr:ring ringstyle="warble" />
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Figure 27: Example Script: Hypothetical Distinctive-Ringing Extension

new attribute regex for the standard address node. In this example, the global namespace is not specified.

13.11 Example: A Complex Example

Finally, Figure 29 is a complex example which shows the sort of


```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <address-switch field="origin" subfield="user"
        xmlns:re="http://www.example.com/regex">
      <address re:regex="(.*.smith|.*.jones)">
         <reject status="reject"
                 reason="I don't want to talk to Smiths or Joneses" />
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Figure 28: Example Script: Hypothetical Regular-Expression Extension

sophisticated behavior which can be achieved by combining CPL nodes. In this case, the user attempts to have his calls reach his desk; if he does not answer within a small amount of time, calls from his boss are forwarded to his mobile phone, and all other calls are directed to voicemail. If the call setup failed, no operation is specified, so the server's default behavior is performed.

14 Security Considerations

The CPL is designed to allow services to be specified in a manner which prevents potentially hostile or mis-configured scripts from launching security attacks, including denial-of-service attacks. Because script runtime is strictly bounded by acyclicity, and because the number of possible script operations are strictly limited, scripts should not be able to inflict damage upon a CPL server.

Because scripts can direct users' telephone calls, the method by which scripts are transmitted from a client to a server MUST be strongly authenticated. Such a method is not specified in this document.

Script servers SHOULD allow server administrators to control the details of what CPL operations are permitted.

15 IANA Considerations

This document registers the MIME type application/cpl+xml. See


```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <subaction id="voicemail">
    <location url="sip:jones@voicemail.example.com">
      <redirect />
    </location>
  </subaction>

  <incoming>
    <location url="sip:jones@phone.example.com">
      <proxy timeout="8">
        <busy>
          <sub ref="voicemail" />
        </busy>
        <noanswer>
          <address-switch field="origin">
            <address is="sip:boss@example.com">
              <location url="tel:+19175551212">
                <proxy />
              </location>
            </address>
            <otherwise>
              <sub ref="voicemail" />
            </otherwise>
          </address-switch>
        </noanswer>
      </proxy>
    </location>
  </incoming>
</cpl>
```

Figure 29: Example Script: A Complex Example

Section 3.2.

16 Acknowledgments

This document was reviewed and commented upon by IETF IP Telephony Working Group. We specifically acknowledge the following people for their help:

The outgoing call screening script was written by Kenny Hom.

Paul E. Jones contributed greatly to the mappings of H.323 addresses.

The text of the time-switch section was taken (lightly modified) from RFC 2445 [12], by Frank Dawson and Derik Stenerson.

We drew a good deal of inspiration, notably the language's lack of Turing-completeness and the syntax of string matching, from the specification of Sieve [19], a language for user filtering of electronic mail messages.

Thomas F. La Porta and Jonathan Rosenberg had many useful discussions, contributions, and suggestions.

A An Algorithm for Resolving Time Switches

The following algorithm resolves, in constant time, whether a given instant falls within a repetition of a time-switch recurrence. Open-source Java code implementing this algorithm is available on the world wide web at <http://www.cs.columbia.edu/~lennox/Cal-Code/>

1. Compute the time of the call, in the timezone of the time switch. (No step after this needs to consider time zones -- all calculations are done using continuously-running standard Gregorian time.)

2. If the call time is earlier than dtstart, fail NOMATCH.

3. If the call time is less than duration after dtstart, succeed MATCH.

4. Determine the smallest unit specified in a byxxx rule or by the freq. Call this the Minimum Unit. Determine the previous instant (before the call time) when all the time units smaller than the minimum unit are the same as those of dtstart. (For all minimum units, the time-of-day must be the same as dtstart. If the minimum unit is a week, the day-of-the-week must be the same as dtstart. If the minimum unit is a month, the day-of-the-month must be the same as dtstart. If the minimum unit is a year, the month and day-of-month must both be the same as dtstart. (Note that this means it may be necessary to roll back more than one minimum unit -- if the minimum unit is a month, then some months do not have a 31st (or 30th or 29th) day; if the minimum unit is a year, then some years do not have a February 29th. In the Gregorian calendar, it is never necessary to roll back more than two months, or eight years (four years between 1904 and 2096).)

Call this instant the Candidate Start Time.

5. If the time between the candidate start time and the call time is more than the duration, fail NOMATCH.

6. If the candidate start time is later than the until parameter of the recurrence, fail NOMATCH.

7. Call the unit of the freq parameter of the recurrence the Frequency Unit. Determine the frequency unit enclosing the Candidate Start Time, and that enclosing dtstart. Calculate the number of frequency units that have passed between these two times. If this is not a multiple of the interval parameter, fail NOMATCH.

8. For every byxxx rule, confirm that the candidate start time matches one of the options specified by that byxxx rule. If not, fail NOMATCH.

9. Succeed MATCH.

B Suggested Usage of CPL with H.323

This appendix gives a suggested usage of CPL with H.323 [2]. Study Group 16 of the ITU, which developed H.323, is proposing to work on official CPL mappings for that protocol. This section is therefore not normative.

B.1 Usage of address-switch with H.323

Address switches are specified in Section 5.1. This section specifies the mapping between H.323 messages and the fields and subfields of address-switches

For H.323, the origin address corresponds to the alias addresses in the sourceAddress field of the Setup-UUIE user-user information element, and to the Q.931 [21] information element "Calling party number." If both fields are present, or if multiple aliases addresses for sourceAddress are present, which one has priority is a matter of local server policy; the server SHOULD use the same resolution as it would use for routing decisions in this case. Similarly, the destination address corresponds to the alias addresses of the destinationAddress field, and to the Q.931 information element "Called party number."

The original-destination address corresponds to the "Redirecting number" Q.931 information element, if it is present; otherwise it is the same as the destination address.

The mapping of H.323 addresses into subfields depends on the type of the alias address. An additional subfield type, alias-type, is defined for H.323 servers, corresponding to the type of the address. Possible values are dialedDigits, h323-ID, url-ID, transportID, email-ID, partyNumber, mobileUIM, and Q.931IE. If future versions of the H.323 specification define additional types of alias addresses, those names MAY also be used.

In versions of H.323 prior to version 4, dialedDigits was known as e164. The two names SHOULD be treated as synonyms.

The value of the address-type subfield for H.323 messages is "h323" unless the alias type is url-ID and the URL scheme is something other than h323; in this case the address-type is the URL scheme, as specified in Section 5.1.1 for SIP.

An H.323-aware CPL server SHOULD map the address subfields from the primary alias used for routing. It MAY also map subfields from other aliases, if subfields in the primary address are not present.

The following mappings are used for H.323 alias types:

dialedDigits, partyNumber, mobileUIM, and Q.931IE: the tel and user subfields are the string of digits, as is the "entire-address" form. The host and port subfields are not present.

url-ID: the same mappings are used as for SIP, in Section 5.1.1.

h323-ID: the user field is the string of characters, as is the "entire-address" form. All other subfields are not present.

email-ID: the user and host subfields are set to the corresponding parts of the e-mail address. The port and tel subfields are not present. The "entire-address" form corresponds to the entire e-mail address.

transportID: if the TransportAddress is of type "ipAddress," "ipSourceRoute," or "ip6Address," the host subfield is set to the "ip" element of the sequence, translated into the standard IPv4 or IPv6 textual representation, and the port subfield is set to the "port" element of the sequence represented in decimal. The tel and user fields are not present. The "entire-address" form is not defined. The representation and mapping of transport addresses is not defined for non-IP addresses.

H.323 version 4 [22] and the Internet-Draft draft-levin-iptel-h323-

url-scheme-00 [23] define a "h323" URI scheme. This appendix defines a mapping for these URIs onto the CPL address-switch subfields, as given in Section 5.1. Neither of these documents has yet been formally published in a final form, so this appendix is non-

normative.

For h323 URIs, the the user, host, and port subfields are set to the corresponding parts of the H.323 URL. The tel subfield is not present. The "entire-address" form corresponds to the entire URI.

This mapping MAY be used both for h323 URIs in an h323 url-ID address alias, and for h323 URIs in SIP messages.

B.2 Usage of string-switch with H.323

For H.323, the string-switch node (see Section 5.2) is used as follows. The field language corresponds to the H.323 UUIE language, translated to the format specified for that field. The field display corresponds to the Q.931 information element of the same name, copied verbatim. The fields subject, organization, and user-agent are not used and are never present.

> The display IE is conventionally used for Caller-ID purposes, so arguably it should be mapped to the display subfield of an address-match with the field originator. However, since a) it is a message-level information element, not an address-level one, and b) the Q.931 specification [21] says only that "[t]he purpose of the Display information element is to supply display information that may be displayed by the user," it seems to be more appropriate to allow it to be matched in a string-switch instead.

B.3 Usage of priority-switch with H.323

All H.323 messages are considered to have priority normal for the purpose of a priority switch (see Section 5.4).

B.4 Usage of location with H.323

Locations in explicit location nodes (Section 6.1) are specified as URLs. Therefore, all locations added in this manner are interpreted as being of alias type url-ID in H.323.

Specifications of other H.323 address alias types will require a CPL extension (see Section 12).

B.5 Usage of lookup with H.323

For location lookup nodes (Section 6.2), the registration lookup source corresponds to the locations registered with the server using RAS messages.

As H.323 currently has no counterpart of SIP caller preferences and callee capabilities, the use and ignore parameters of the lookup node

are ignored.

B.6 Usage of remove-location with H.323

For location removal nodes (Section 6.3), only literal URLs can be removed. No URL patterns are defined.

As H.323 currently has no counterpart of SIP caller preferences and callee capabilities, the param and value parameters of the remove-location node are ignored.

C The XML DTD for CPL

This section includes a full DTD describing the XML syntax of the CPL. Every script submitted to a CPL server SHOULD comply with this DTD. However, CPL servers MAY allow minor variations from it, particularly in the ordering of the outputs of nodes. Note that compliance with this DTD is not a sufficient condition for correctness of a CPL script, as many of the conditions described above are not expressible in DTD syntax.


```
<?xml version="1.0" encoding="US-ASCII" ?>

<!--
    Draft DTD for CPL, corresponding to
    draft-ietf-iptel-cpl-01.
-->

<!-- Nodes. -->
<!-- Switch nodes -->
<!ENTITY % Switch 'address-switch|string-switch|time-switch|
                   priority-switch' >
```

```
<!-- Location nodes -->
<!ENTITY % Location 'location|lookup|remove-location' >

<!-- Signalling action nodes -->
<!ENTITY % SignallingAction 'proxy|redirect|reject' >

<!-- Other actions -->
<!ENTITY % OtherAction 'mail|log' >

<!-- Links to subactions -->
<!ENTITY % Sub 'sub' >

<!-- Nodes are one of the above four categories, or a subaction.
     This entity (macro) describes the contents of an output.
     Note that a node can be empty, implying default action. -->
<!ENTITY % Node     '(%Location;|%Switch;|%SignallingAction;|
                      %OtherAction;|%Sub;)?' >


<!-- Switches: choices a CPL script can make. -->

<!-- All switches can have an 'otherwise' output. -->
<!ELEMENT otherwise ( %Node; ) >

<!-- All switches can have a 'not-present' output. -->
<!ELEMENT not-present ( %Node; ) >

<!-- Address-switch makes choices based on addresses. -->
<!ELEMENT address-switch ( (address|not-present)+, otherwise? ) >
<!-- <not-present> must appear at most once -->
<!ATTLIST address-switch
   field         CDATA    #REQUIRED
   subfield      CDATA    #IMPLIED
>
```



```
<!ELEMENT address ( %Node; ) >

<!ATTLIST address
   is            CDATA    #IMPLIED
   contains      CDATA    #IMPLIED
   subdomain-of  CDATA    #IMPLIED
> <!-- Exactly one of these three attributes must appear -->


<!-- String-switch makes choices based on strings. -->

<!ELEMENT string-switch ( (string|not-present)+, otherwise? ) >
<!-- <not-present> must appear at most once -->
<!ATTLIST string-switch
   field         CDATA    #REQUIRED
>
```

```
<!ELEMENT string ( %Node; ) >
<!ATTLIST string
   is             CDATA    #IMPLIED
   contains       CDATA    #IMPLIED
>  <!-- Exactly one of these two attributes must appear -->

<!-- Time-switch makes choices based on the current time. -->

<!ELEMENT time-switch ( (time|not-present)+, otherwise? ) >
<!ATTLIST time-switch
   tzid           CDATA    #IMPLIED
   tzurl          CDATA    #IMPLIED
>

<!ELEMENT time ( %Node; ) >

<!-- Exactly one of the two attributes "dtend" and "duration"
     must occur. -->
<!-- The value of "freq" is (daily|weekly|monthly|yearly).  It is
        case-insensitive, so it is not given as a DTD switch. -->
<!-- None of the attributes following freq are meaningful unless freq
         appears. -->
<!-- The value of "wkst" is (MO|TU|WE|TH|FR|SA|SU).  It is
        case-insensitive, so it is not given as a DTD switch. -->
<!ATTLIST time
   dtstart        CDATA  #REQUIRED
   dtend          CDATA  #IMPLIED
   duration       CDATA  #IMPLIED
   freq           CDATA  #IMPLIED
   until          CDATA  #IMPLIED
   interval       CDATA  "1"
```



```
   byday          CDATA  #IMPLIED
   bymonthday     CDATA  #IMPLIED
   byyearday      CDATA  #IMPLIED
   byweekno       CDATA  #IMPLIED
   bymonth        CDATA  #IMPLIED
   wkst           CDATA  "MO"
>


<!-- Priority-switch makes choices based on message priority. -->

<!ELEMENT priority-switch ( (priority|not-present)+, otherwise? ) >
<!-- <not-present> must appear at most once -->

<!ENTITY % PriorityVal '(emergency|urgent|normal|non-urgent)' >

<!ELEMENT priority ( %Node; ) >

<!-- Exactly one of these three attributes must appear -->
<!ATTLIST priority
```

```
   less          %PriorityVal;  #IMPLIED
   greater       %PriorityVal;  #IMPLIED
   equal         CDATA          #IMPLIED
>


<!-- Locations: ways to specify the location a subsequent action
     (proxy, redirect) will attempt to contact. -->

<!ENTITY % Clear  'clear (yes|no) "no"' >

<!ELEMENT location ( %Node; ) >
<!ATTLIST location
   url           CDATA    #REQUIRED
   priority      CDATA    #IMPLIED
   %Clear;
>

<!ELEMENT lookup ( success,notfound?,failure? ) >
<!ATTLIST lookup
  source         CDATA     #REQUIRED
  timeout        CDATA     "30"
  use            CDATA     #IMPLIED
  ignore         CDATA     #IMPLIED
  %Clear;
>

<!ELEMENT success  ( %Node; ) >
```



```
<!ELEMENT notfound ( %Node; ) >
<!ELEMENT failure ( %Node; ) >

<!ELEMENT remove-location ( %Node; ) >
<!ATTLIST remove-location
   param         CDATA    #IMPLIED
   value         CDATA    #IMPLIED
   location      CDATA    #IMPLIED
>


<!-- Signalling Actions: call-signalling actions the script can
     take. -->

<!ELEMENT proxy ( busy?,noanswer?,redirection?,failure?,default? ) >

<!-- The default value of timeout is "20" if the <noanswer> output
     exists. -->
<!ATTLIST proxy
   timeout       CDATA    #IMPLIED
   recurse       (yes|no) "yes"
   ordering      CDATA    "parallel"
>
```

```
<!ELEMENT busy ( %Node; ) >
<!ELEMENT noanswer ( %Node; ) >
<!ELEMENT redirection ( %Node; ) >
<!-- "failure" repeats from lookup, above. -->
<!ELEMENT default ( %Node; ) >

<!ELEMENT redirect EMPTY >
<!ATTLIST redirect
   permanent     (yes|no) "no"
>


<!-- Statuses we can return -->

<!ELEMENT reject EMPTY >
<!-- The value of "status" is (busy|notfound|reject|error), or a SIP
     4xx-6xx status. -->
<!ATTLIST reject
   status        CDATA    #REQUIRED
   reason        CDATA    #IMPLIED
>

<!-- Non-signalling actions: actions that don't affect the call -->
```



```
<!ELEMENT mail ( %Node; ) >
<!ATTLIST mail
   url           CDATA    #REQUIRED
>

<!ELEMENT log ( %Node; ) >
<!ATTLIST log
   name          CDATA    #IMPLIED
   comment       CDATA    #IMPLIED
>


<!-- Calls to subactions. -->

<!ELEMENT sub EMPTY >
<!ATTLIST sub
   ref           IDREF    #REQUIRED
>


<!-- Ancillary data -->

<!ENTITY % Ancillary 'ancillary?' >

<!ELEMENT ancillary EMPTY >


<!-- Subactions -->
```

```
<!ENTITY % Subactions 'subaction*' >

<!ELEMENT subaction ( %Node; )>
<!ATTLIST subaction
   id             ID       #REQUIRED
>


<!-- Top-level actions -->

<!ENTITY % TopLevelActions 'outgoing?,incoming?' >

<!ELEMENT outgoing ( %Node; )>

<!ELEMENT incoming ( %Node; )>

<!-- The top-level element of the script. -->

<!ELEMENT cpl  ( %Ancillary;,%Subactions;,%TopLevelActions; ) >
```


D Changes from Earlier Versions

[Note to RFC Editor: please remove this appendix before publication as an RFC.]

D.1 Changes from Draft -03

The changebars in the Postscript and PDF versions of this document indicate significant changes from this version.

- Removed an obsolete reference to a usage in examples which wasn't actually used anywhere.
- Added forward references to remove-location, mail and log, as well as location, in the XML syntax as examples of nodes that don't have explicit output tags.
- Made the usage of some terminology more consistent: "output" vs. "next node"; "action" vs. "operation" vs. "behavior"; "sub-actions" and "subactions"; "other operations" and "non-call operations" and "non-signalling operations"; "meta-information" and "ancillary information."
- The tel subfield of addresses which come from sip URIs should have its visual separators stripped.
- The default value of the priority value of the location node is 1.0.
- Corrected the media type of a set of URIs to text/uri-list, and added a reference to it.

- Added some wording clarifying how URI-based lookup queries work.

- Corrected the syntax of duration parameter in the examples.

- Performed some pre-RFC textual cleanups (e.g. removing the reference to the Internet-Draft URL from the XML namespace identifier).

- Re-worded text in the description of the Ancillary tag which implied that information could be placed in that node in the base CPL specification. Clarified that the tag is for use by extensions only.

- Expunged some references to sub-daily recurrences which had

accidentally been left in the text.

- Updated bibliography to refer to the latest versions of the cited documents.

- Fixed a number of typographical errors.

D.2 Changes from Draft -02

- Reduced time-switches from the full iCal recurrence to an iCal subset. Added an appendix giving an algorithm to resolve time-switches.

- Added the extension mechanism.

- Made explicit how each node is dependent on protocol handling. Separated out protocol-specific information -- for SIP in subsections of the main text, for H.323 in a non-normative appendix.

- Clarified some address mapping rules for H.323.

- Corrected the name of the "Redirecting number" in Q.931.

- Clarified that address matching on the password subfield is case-sensitive.

- Added a recommendation that TZID labels follow the usage of the Olson database.

- Added the priority parameter to location nodes.

- Added the default output to the proxy node.

- Made the meaning of the proxy node's outputs explicit.

- Added suggested content for the e-mail generated by mail nodes.
- Pointed out that "&" must be escaped in XML (this is relevant for mailto URIs).
- Pointed out that log names are logical names, and should not be interpreted as verbatim filenames.
- Added some examples.
- Clarified some wording.

- Fixed some typographical errors.

D.3 Changes from Draft -01

- Completely re-wrote changes to time switches: they are now based on iCal rather than on crontab.
- Timezone references are now defined within time switches rather than in the ancillary section. The ancillary section is now empty, but still defined for future use. To facilitate this, an explicit ancillary tag was added.
- Added XML document type identifiers (the public identifier and the namespace), and MIME registration information.
- Clarified that the not-present output can appear anywhere in a switch.
- Re-wrote H.323 address mappings. Added the alias-type subfield for H.323 addresses.
- Added the language and display string switch fields.
- Clarified why useless not-present outputs can appear in time and priority switches.
- Added the clear parameter to location and lookup nodes. (It had been in the DTD previously, but not in the text.)
- Weakened support for non-validating scripts from SHOULD to MAY, to allow the use of validating XML parsers.
- Added redirection output of proxy nodes.
- Clarified some aspects of how proxy nodes handle the location set.
- Added permanent parameter of redirect nodes.
- Add example script for outgoing call screening (from Kenny

Hom)

- Updated example scripts to use the public identifier.
- Add omitted tag to example script for call forward busy/no answer
- Clarified in introduction that this document mainly deals with

servers.

- Updated reference to RFC 2824 now that it has been published.
- Added explanatory text to the introduction to types of nodes.
- Numerous minor clarifications and wording changes.
- Fixed copy-and-paste errors, typos.

D.4 Changes from Draft -00

- Added high-level structure; script doesn't just start at a first action.
- Added a section giving a high-level explanation of the location model.
- Added informal syntax specifications for each tag so people don't have to try to understand a DTD to figure out the syntax.
- Added subactions, replacing the old link tags. Links were far too reminiscent of gotos for everyone's taste.
- Added ancillary information section, and timezone support.
- Added not-present switch output.
- Added address switches.
- Made case-insensitive string matching locale-independent.
- Added priority switch.
- Deleted "Other switches" section. None seem to be needed.
- Unified url and source parameters of lookup.
- Added caller prefs to lookup.
- Added location filtering.
- Eliminated "clear" parameter of location setting. Instead,

proxy "eats" locations it has used.

o Added recurse and ordering parameters to proxy.

o Added default value of timeout for proxy.

o Renamed response to reject.

o Changed notify to mail, and simplified it.

o Simplified log, eliminating its failure output.

o Added description of default actions at various times during script processing.

o Updated examples for these changes.

o Updated DTD to reflect new syntax.

E Authors' Addresses

Jonathan Lennox
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA
electronic mail: lennox@cs.columbia.edu

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA
electronic mail: schulzrinne@cs.columbia.edu

F Bibliography

[1] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[2] International Telecommunication Union, "Packet based multimedia communication systems," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[3] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible markup language (XML) 1.0 (second edition)," W3C Recommendation REC-xml-20001006, World Wide Web Consortium (W3C), Oct. 2000. Available at http://www.w3.org/XML/.

[4] J. Lennox and H. Schulzrinne, "Call processing language framework and requirements," Request for Comments 2824, Internet Engineering Task Force, May 2000.

[5] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments 2119, Internet Engineering Task Force, Mar. 1997.

[6] D. Raggett, A. Le Hors, and I. Jacobs, "HTML 4.01 specification," W3C Recommendation REC-html401-19991224, World Wide Web Consortium (W3C), Dec. 1999. Available at http://www.w3.org/TR/html4/.

[7] ISO (International Organization for Standardization), "Information processing -- text and office systems -- standard generalized markup language (SGML)," ISO Standard ISO 8879:1986(E), International Organization for Standardization, Geneva, Switzerland, Oct. 1986.

[8] M. Murata, S. S. Laurent, and D. Kohn, "XML media types," Request for Comments YYYY, Internet Engineering Task Force, Sept. 2000. [Draft draft-murata-xml-09.txt, approved for Proposed Standard. RFC Editor: please fill in appropriate bibliographic information.].

[9] H. Alvestrand, "Tags for the identification of languages," Request for Comments 1766, Internet Engineering Task Force, Mar. 1995.

[10] M. Davis and M. Duerst, "Unicode normalization forms," Unicode Technical Report 15, Unicode Consortium, Aug. 2000. Revision 19; part of Unicode 3.0.1. Available at http://www.unicode.org/unicode/reports/tr15/.

[11] M. Davis, "Case mappings," Unicode Technical Report 21, Unicode Consortium, Oct. 2000. Revision 4.3. Available at http://www.unicode.org/unicode/reports/tr21/.

[12] F. Dawson and D. Stenerson, "Internet calendaring and scheduling core object specification (icalendar)," Request for Comments 2445, Internet Engineering Task Force, Nov. 1998.

[13] P. Eggert, "Sources for time zone and daylight saving time data." Available at http://www.twinsun.com/tz/tz-link.htm.

[14] ISO (International Organization for Standardization), "Data elements and interchange formats -- information interchange -- representation of dates and times," ISO Standard ISO 8601:1988(E), International Organization for Standardization, Geneva, Switzerland, June 1986.

[15] M. Mealling and R. Daniel, "URI resolution services necessary for URN resolution," Request for Comments 2483, Internet Engineering Task Force, Jan. 1999.

[16] H. Schulzrinne and J. Rosenberg, "SIP caller preferences and callee capabilities," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[17] S. DeRose, E. Maler, D. Orchard, and B. Trafford, "XML linking language (XLink) version 1.0," W3C Candidate Recommendation CR-xlink-20000703, World Wide Web Consortium (W3C), July 2000. Available at http://www.w3.org/TR/xlink/.

[18] T. Bray, D. Hollander, and A. Layman, "Namespaces in XML," W3C Recommendation REC-xml-names-19900114, World Wide Web Consortium (W3C), Jan. 1999. Available at http://www.w3.org/TR/REC-xml-names/.

[19] T. Showalter, "Sieve: A mail filtering language," Request for Comments YYYY, Internet Engineering Task Force, Aug. 2000. [Draft draft-showalter-sieve-12.txt, approved for Proposed Standard. RFC Editor: please fill in appropriate bibliographic information.].

[20] D. C. Fallside, "XML schema part 0: Primer," W3C Candidate Recommendation CR-xmlschema-0-20001024, World Wide Web Consortium (W3C), Oct. 2000. Available at http://www.w3.org/TR/xmlschema-0/.

[21] International Telecommunication Union, "Digital subscriber signalling system no. 1 (dss 1) - isdn user-network interface layer 3 specification for basic call control," Recommendation Q.931, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1993.

[22] International Telecommunication Union, "Packet based multimedia communication systems," Recommendation H.323 Draft v4, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, July 2000. To be published November 2000.

[23] O. Levin, "H.323 URL scheme definition," Internet Draft, Internet Engineering Task Force, Aug. 2000. Work in progress.

Full Copyright Statement

Table of Contents

Network Working Group
Request for Comments: 2824
Category: Informational

J. Lennox
H. Schulzrinne
Columbia University
May 2000

Call Processing Language Framework and Requirements

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

A large number of the services we wish to make possible for Internet telephony require fairly elaborate combinations of signalling operations, often in network devices, to complete. We want a simple and standardized way to create such services to make them easier to implement and deploy. This document describes an architectural framework for such a mechanism, which we call a call processing language. It also outlines requirements for such a language.

Table of Contents

1 Introduction

Recently, several protocols have been created to allow telephone calls to be made over IP networks, notably SIP [1] and H.323 [2]. These emerging standards have opened up the possibility of a broad and dramatic decentralization of the provisioning of telephone services so they can be under the user's control.

Many Internet telephony services can, and should, be implemented entirely on end devices. Multi-party calls, for instance, or call waiting alert tones, or camp-on services, depend heavily on end-system state and on the specific content of media streams, information which often is only available to the end system. A variety of services, however -- those involving user location, call distribution, behavior when end systems are busy, and the like -- are independent of a particular end device, or need to be operational even when an end device is unavailable. These services are still best located in a network device, rather than in an end system.

Traditionally, network-based services have been created only by service providers. Service creation typically involved using proprietary or restricted tools, and there was little range for customization or enhancement by end users. In the Internet environment, however, this changes. Global connectivity and open protocols allow end users or third parties to design and implement new or customized services, and to deploy and modify their services dynamically without requiring a service provider to act as an intermediary.

A number of Internet applications have such customization environments -- the web has CGI [3], for instance, and e-mail has Sieve [4] or procmail. To create such an open customization environment for Internet telephony, we need a standardized, safe way

for these new service creators to describe the desired behavior of network servers.

This document describes an architecture in which network devices respond to call signalling events by triggering user-created programs written in a simple, static, non-expressively-complete language. We call this language a call processing language.

The development of this document has been substantially informed by the development of a particular call processing language, as described in [5]. In general, when this document refers to "a call processing language," it is referring to a generic language that fills this role; "the call processing language" or "the CPL" refers to this particular language.

2 Terminology

In this section we define some of the terminology used in this document.

SIP [1] terminology used includes:

invitation: The initial INVITE request of a SIP transaction, by which one party initiates a call with another.

redirect server: A SIP device which responds to invitations and other requests by informing the request originator of an alternate address to which the request should be sent.

proxy server: A SIP device which receives invitations and other requests, and forwards them to other SIP devices. It then receives the responses to the requests it forwarded, and forwards them back to the sender of the initial request.

user agent: A SIP device which creates and receives requests, so as to set up or otherwise affect the state of a call. This may be, for example, a telephone or a voicemail system.

user agent client: The portion of a user agent which initiates requests.

user agent server: The portion of a user agent which responds to requests.

H.323 [2] terminology used includes:

terminal: An H.323 device which originates and receives calls, and their associated media.

gatekeeper: An H.323 entity on the network that provides address translation and controls access to the network for H.323 terminals and other endpoints. The gatekeeper may also

provide other services to the endpoints such as bandwidth management and locating gateways.

gateway: A device which translates calls between an H.323 network and another network, typically the public-switched telephone network.

RAS: The Registration, Admission and Status messages communicated between two H.323 entities, for example between an endpoint and a gatekeeper.

General terminology used in this document includes:

user location: The process by which an Internet telephony device determines where a user named by a particular address can be found.

CPL: A Call Processing Language, a simple language to describe how Internet telephony call invitations should be processed.

script: A particular instance of a CPL, describing a particular set of services desired.

end system: A device from which and to which calls are established. It creates and receives the call's media (audio, video, or the like). This may be a SIP user agent or an H.323 terminal.

signalling server: A device which handles the routing of call invitations. It does not process or interact with the media of a call. It may be a SIP proxy or redirect server, or an H.323 gatekeeper.

3 Example services

To motivate the subsequent discussion, this section gives some specific examples of services which we want users to be able to create programmatically. Note that some of these examples are deliberately somewhat complicated, so as to demonstrate the level of decision logic that should be possible.

- Call forward on busy/no answer

  When a new call comes in, the call should ring at the user's desk telephone. If it is busy, the call should always be redirected to the user's voicemail box. If, instead, there's no answer after four rings, it should also be redirected to his or her voicemail, unless it's from a supervisor, in which case it should be proxied to the user's cell phone if it is currently registered.

- Information address

A company advertises a general "information" address for prospective customers. When a call comes in to this address, if it's currently working hours, the caller should be given a list of the people currently willing to accept general information calls. If it's outside of working hours, the caller should get a webpage indicating what times they can call.

- Intelligent user location

  When a call comes in, the list of locations where the user has registered should be consulted. Depending on the type of call (work, personal, etc.), the call should ring at an appropriate subset of the registered locations, depending on information in the registrations. If the user picks up from more than one station, the pick-ups should be reported back separately to the calling party.

- Intelligent user location with media knowledge

  When a call comes in, the call should be proxied to the station the user has registered from whose media capabilities best match those specified in the call request. If the user does not pick up from that station within four rings, the call should be proxied to the other stations from which he or she has registered, sequentially, in order of decreasing closeness of match.

- Client billing allocation -- lawyer's office

  When a call comes in, the calling address is correlated with the corresponding client, and client's name, address, and the time of the call is logged. If no corresponding client is found, the call is forwarded to the lawyer's secretary.

4 Usage scenarios

A CPL would be useful for implementing services in a number of different scenarios.

- Script creation by end user

  In the most direct approach for creating a service with a CPL, an end user simply creates a script describing their service. He or she simply decides what service he or she wants, describes it using a CPL script, and then uploads it to a server.

- Third party outsourcing

  Because a CPL is a standardized language, it can also be used

to allow third parties to create or customize services for clients. These scripts can then be run on servers owned by the end user or the user's service provider.

- Administrator service definition

  A CPL can also be used by server administrators to create simple services or describe policy for servers they control. If a server is implementing CPL services in any case, extending the service architecture to allow administrators as well as users to create scripts is a simple extension.

- Web middleware

  Finally, there have been a number of proposals for service creation or customization using web interfaces. A CPL could be used as the back-end to such environments: a web application could create a CPL script on behalf of a user, and the telephony server could then implement the services without either component having to be aware of the specifics of the other.

5 CPL creation

There are also a number of means by which CPL scripts could be created. Like HTML, which can be created in a number of different manners, we envision multiple creation styles for a CPL script.

- Hand authoring

  Most directly, CPL scripts can be created by hand, by knowledgeable users. The CPL described in [5] has a text format with an uncomplicated syntax, so hand authoring will be straightforward.

- Automated scripts

  CPL features can be created by automated means, such as in the example of the web middleware described in the previous section. With a simple, text-based syntax, standard text-processing languages will be able to create and edit CPL scripts easily.

- GUI tools

  Finally, users will be able to use GUI tools to create and edit CPL scripts. We expect that most average-experience users will take this approach once the CPL gains popularity. The CPL will

be designed with this application in mind, so that the full expressive power of scripts can be represented simply and straightforwardly in a graphical manner.

6 Network model

The Call Processing Language operates on a generalized model of an Internet telephony network. While the details of various protocols differ, on an abstract level all major Internet telephony architectures are sufficiently similar that their major features can be described commonly. This document generally uses SIP terminology, as its authors' experience has mainly been with that protocol.

6.1 Model components

In the Call Processing Language's network model, an Internet telephony network contains two types of components.

6.1.1 End systems

End systems are devices which originate and/or receive signalling information and media. These include simple and complex telephone devices, PC telephony clients, and automated voice systems. The CPL abstracts away the details of the capabilities of these devices. An end system can originate a call; and it can accept, reject, or forward incoming calls. The details of this process (ringing, multi-line telephones, and so forth) are not important for the CPL.

For the purposes of the CPL, gateways -- for example, a device which connects calls between an IP telephony network and the PSTN -- are also considered to be end systems. Other devices, such as mixers or firewalls, are not directly dealt with by the CPL, and they will not be discussed here.

6.1.2 Signalling servers

Signalling servers are devices which relay or control signalling information. In SIP, they are proxy servers, redirect servers, or registrars; in H.323, they are gatekeepers.

Signalling servers can perform three types of actions on call setup information. They can:

proxy it: forward it on to one or more other network or end systems, returning one of the responses received.

redirect it: return a response informing the sending system of a different address to which it should send the request.

reject it: inform the sending system that the setup request could not be completed.

RFC 2543 [1] has illustrations of proxy and redirect functionality. End systems may also be able to perform some of these actions: almost certainly rejection, and possibly redirection.

Signalling servers also normally maintain information about user location. Whether by means of registrations (SIP REGISTER or H.323 RAS messages), static configuration, or dynamic searches, signalling servers must have some means by which they can determine where a user is currently located, in order to make intelligent choices about their proxying or redirection behavior.

Signalling servers are also usually able to keep logs of transactions that pass through them, and to send e-mail to destinations on the Internet, under programmatic control.

6.2 Component interactions

When an end system places a call, the call establishment request can proceed by a variety of routes through components of the network. To begin with, the originating end system must decide where to send its requests. There are two possibilities here: the originator may be configured so that all its requests go to a single local server; or it may resolve the destination address to locate a remote signalling server or end system to which it can send the request directly.

Once the request arrives at a signalling server, that server uses its user location database, its local policy, DNS resolution, or other methods, to determine the next signalling server or end system to which the request should be sent. A request may pass through any number of signalling servers: from zero (in the case when end systems communicate directly) to, in principle, every server on the network. What's more, any end system or signalling server can (in principle) receive requests from or send them to any other.

For example, in figure 1, there are two paths the call establishment request information may take. For Route 1, the originator knows only a user address for the user it is trying to contact, and it is configured to send outgoing calls through a local outgoing proxy server. Therefore, it forwards the request to its local server, which finds the server of record for that address, and forwards it on to that server.

In this case, the organization the destination user belongs to uses a multi-stage setup to find users. The corporate server identifies which department a user is part of, then forwards the request to the appropriate departmental server, which actually locates the user. (This is similar to the way e-mail forwarding is often configured.) The response to the request will travel back along the same path.

For Route 2, however, the originator knows the specific device address it is trying to contact, and it is not configured to use a local outgoing proxy. In this case, the originator can directly contact the destination without having to communicate with any

network servers at all.

We see, then, that in Internet telephony signalling servers cannot in general know the state of end systems they "control," since signalling information may have bypassed them. This architectural limitation implies a number of restrictions on how some services can be implemented. For instance, a network system cannot reliably know if an end system is currently busy or not; a call may have been placed to the end system without traversing that network system. Thus, signalling messages must explicitly travel to end systems to find out their state; in the example, the end system must explicitly return a "busy" indication.

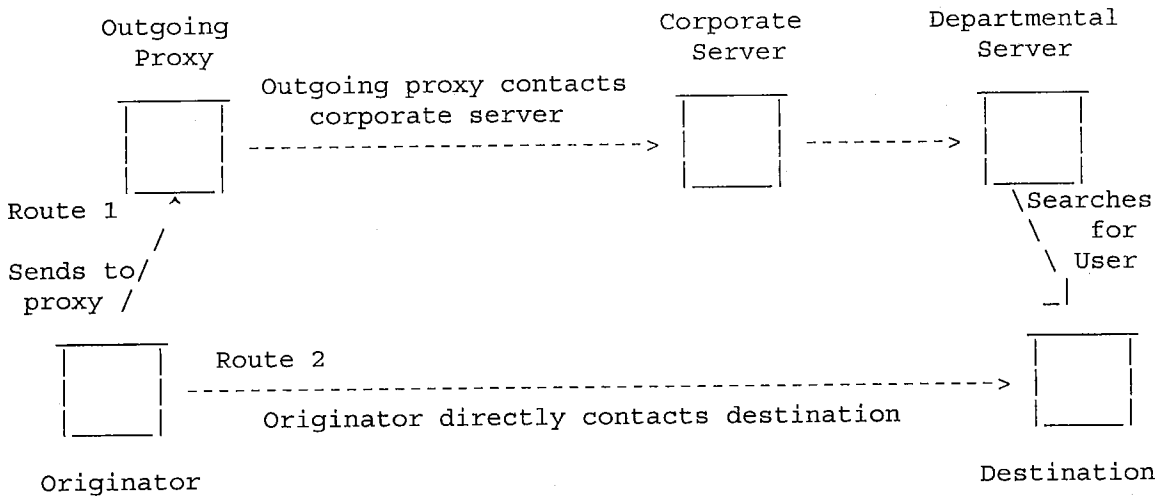


Figure 1: Possible paths of call setup messages

7 Interaction of CPL with network model

7.1 What a script does

A CPL script runs in a signalling server, and controls that system's proxy, redirect, or rejection actions for the set-up of a particular call. It does not attempt to coordinate the behavior of multiple signalling servers, or to describe features on a "Global Functional Plane" as in the Intelligent Network architecture [6].

More specifically, a script replaces the user location functionality of a signalling server. As described in section 6.1.2, a signalling server typically maintains a database of locations where a user can

be reached; it makes its proxy, redirect, and rejection decisions based on the contents of that database. A CPL script replaces this basic database lookup functionality; it takes the registration information, the specifics of a call request, and other external information it wants to reference, and chooses the signalling actions to perform.

Abstractly, a script can be considered as a list of condition/action pairs; if some attribute of the registration, request, and external information matches a given condition, then the corresponding action (or more properly set of actions) is taken. In some circumstances, additional actions can be taken based on the consequences of the first action and additional conditions. If no condition matches the invitation, the signalling server's standard action -- its location database lookup, for example -- is taken.

7.2 Which script is executed

CPL scripts are usually associated with a particular Internet telephony address. When a call establishment request arrives at a signalling server which is a CPL server, that server associates the source and destination addresses specified in the request with its database of CPL scripts; if one matches, the corresponding script is executed.

Once the script has executed, if it has chosen to perform a proxy action, a new Internet telephony address will result as the destination of that proxying. Once this has occurred, the server again checks its database of scripts to see if any of them are associated with the new address; if one is, that script as well is executed (assuming that a script has not attempted to proxy to an address which the server has already tried). For more details of this recursion process, and a description of what happens when a server has scripts that correspond both to a scripts origination address and its destination address, see section 9.2.

In general, in an Internet telephony network, an address will denote one of two things: either a user, or a device. A user address refers to a particular individual, for example sip:joe@example.com, regardless of where that user actually is or what kind of device he or she is using. A device address, by contrast, refers to a particular physical device, such as sip:x26063@phones.example.com. Other, intermediate sorts of addresses are also possible, and have some use (such as an address for "my cell phone, wherever it currently happens to be registered"), but we expect them to be less common. A CPL script is agnostic to the type of address it is associated with; while scripts associated with user addresses are probably the most useful for most services, there is no reason that a script could not be associated with any other type of address as well. The recursion process described above allows scripts to be associated with several of a user's addresses; thus, a user script could specify an action "try me at my cell phone," whereas a device

script could say "I don't want to accept cell phone calls while I'm out of my home area."

It is also possible for a CPL script to be associated not with one specific Internet telephony address, but rather with all addresses handled by a signalling server, or a large set of them. For instance, an administrator might configure a system to prevent calls from or to a list of banned incoming or outgoing addresses; these should presumably be configured for everyone, but users should still to be able to have their own custom scripts as well. Exactly when such

scripts should be executed in the recursion process depends on the precise nature of the administrative script. See section 9.2 for further discussion of this.

7.3 Where a script runs

Users can have CPL scripts on any network server which their call establishment requests pass through and with which they have a trust relationship. For instance, in the example in figure 1, the originating user could have a script on the outgoing proxy, and the destination user could have scripts on both the corporate server and the departmental server. These scripts would typically perform different functions, related to the role of the server on which they reside; a script on the corporate-wide server could be used to customize which department the user wishes to be found at, for instance, whereas a script at the departmental server could be used for more fine-grained location customization. Some services, such as filtering out unwanted calls, could be located at either server. See section 9.3 for some implications of a scenario like this.

This model does not specify the means by which users locate a CPL-capable network server. In general, this will be through the same means by which they locate a local Internet telephony server to register themselves with; this may be through manual configuration, or through automated means such as the Service Location Protocol [7]. It has been proposed that automated means of locating such servers should include a field to indicate whether the server allows users to upload CPLs.

8 Creation and transport of a call processing language script

Users create call processing language scripts, typically on end devices, and transmit them through the network to signalling servers. Scripts persist in signalling servers until changed or deleted, unless they are specifically given an expiration time; a network system which supports CPL scripting will need stable storage.

The end device on which the user creates the CPL script need not bear any relationship to the end devices to which calls are actually placed. For example, a CPL script might be created on a PC, whereas

calls might be intended to be received on a simple audio-only telephone. Indeed, the device on which the script is created may not be an "end device" in the sense described in section 6.1.1 at all; for instance, a user could create and upload a CPL script from a non-multimedia-capable web terminal.

The CPL also might not necessarily be created on a device near either the end device or the signalling server in network terms. For example, a user might decide to forward his or her calls to a remote location only after arriving at that location.

The exact means by which the end device transmits the script to the server remains to be determined; it is likely that many solutions will be able to co-exist. This method will need to be authenticated in almost all cases. The methods that have been suggested include web file upload, SIP REGISTER message payloads, remote method invocation, SNMP, ACAP, LDAP, and remote file systems such as NFS.

Users can also retrieve their current script from the network to an end system so it can be edited. The signalling server should also be able to report errors related to the script to the user, both static errors that could be detected at upload time, and any run-time errors that occur.

If a user has trust relationships with multiple signalling servers (as discussed in section 7.3), the user may choose to upload scripts to any or all of those servers. These scripts can be entirely independent.

9 Feature interaction behavior

Feature interaction is the term used in telephony systems when two or more requested features produce ambiguous or conflicting behavior [8]. Feature interaction issues for features implemented with a call processing language can be roughly divided into three categories: feature-to-feature in one server, script-to-script in one server, and server-to-server.

9.1 Feature-to-feature interactions

Due to the explicit nature of event conditions discussed in the previous section, feature-to-feature interaction is not likely to be a problem in a call processing language environment. Whereas a subscriber to traditional telephone features might unthinkingly subscribe to both "call waiting" and "call forward on busy," a user creating a CPL script would only be able to trigger one action in response to the condition "a call arrives while the line is busy." Given a good user interface for creation, or a CPL server which can check for unreachable code in an uploaded script, contradictory condition/action pairs can be avoided.

9.2 Script-to-script interactions

Script-to-script interactions arise when a server invokes multiple scripts for a single call, as described in section 7.2. This can occur in a number of cases: if both the call originator and the destination have scripts specified on a single server; if a script forwards a request to another address which also has a script; or if an administrative script is specified as well as a user's individual script.

The solution to this interaction is to determine an ordering among the scripts to be executed. In this ordering, the "first" script is executed first; if this script allows or permits the call to be proxied, the script corresponding to the next address is executed. When the first script says to forward the request to some other address, those actions are considered as new requests which arrive at the second script. When the second script sends back a final response, that response arrives at the first script in the same manner as if a request arrived over the network. Note that in some cases, forwarding can be recursive; a CPL server must be careful to prevent forwarding loops.

Abstractly, this can be viewed as equivalent to having each script execute on a separate signalling server. Since the CPL architecture is designed to allow scripts to be executed on multiple signalling servers in the course of locating a user, we can conceptually transform script-to-script interactions into the server-to-server interactions described in the next section, reducing the number of types of interactions we need to concern ourselves with.

The question, then, is to determine the correct ordering of the scripts. For the case of a script forwarding to an address which also has a script, the ordering is obvious; the other two cases are somewhat more subtle. When both originator and destination scripts exist, the originator's script should be executed before the destination script; this allows the originator to perform address translation, call filtering, etc., before a destination address is determined and a corresponding script is chosen.

Even more complicated is the case of the ordering of administrative scripts. Many administrative scripts, such as ones that restrict source and destination addresses, need to be run after originator scripts, but before destination scripts, to avoid a user's script evading administrative restrictions through clever forwarding; however, others, such as a global address book translation function, would need to be run earlier or later. Servers which allow

administrative scripts to be run will need to allow the administrator to configure when in the script execution process a particular administrative script should fall.

9.3 Server-to-server interactions

The third case of feature interactions, server-to-server interactions, is the most complex of these three. The canonical example of this type of interaction is the combination of Originating Call Screening and Call Forwarding: a user (or administrator) may wish to prevent calls from being placed to a particular address, but the local script has no way of knowing if a call placed to some other, legitimate address will be proxied, by a remote server, to the banned address. This type of problem is unsolvable in an administratively heterogeneous network, even a "lightly" heterogeneous network such as current telephone systems. CPL does not claim to solve it, but the problem is not any worse for CPL scripts than for any other means of deploying services.

Another class of server-to-server interactions are best resolved by the underlying signalling protocol, since they can arise whether the signalling servers are being controlled by a call processing language or by some entirely different means. One example of this is forwarding loops, where user X may have calls forwarded to Y, who has calls forwarded back to X. SIP has a mechanism to detect such loops. A call processing language server thus does not need to define any special mechanisms to prevent such occurrences; it should, however, be possible to trigger a different set of call processing actions in the event that a loop is detected, and/or to report back an error to the owner of the script through some standardized run-time error reporting mechanism.

9.4 Signalling ambiguity

As an aside, [8] discusses a fourth type of feature interaction for traditional telephone networks, signalling ambiguity. This can arise when several features overload the same operation in the limited signal path from an end station to the network: for example, flashing the switch-hook can mean both "add a party to a three-way call" and "switch to call waiting." Because of the explicit nature of signalling in both the Internet telephony protocols discussed here, this issue does not arise.

10 Relationship with existing languages

This document's description of the CPL as a "language" is not intended to imply that a new language necessarily needs to be implemented from scratch. A server could potentially implement all

the functionality described here as a library or set of extensions for an existing language; Java, or the various freely-available scripting languages (Tcl, Perl, Python, Guile), are obvious possibilities.

However, there are motivations for creating a new language. All the existing languages are, naturally, expressively complete; this has two inherent disadvantages. The first is that any function implemented in them can take an arbitrarily long time, use an arbitrarily large amount of memory, and may never terminate. For call processing, this sort of resource usage is probably not necessary, and as described in section 12.1, may in fact be undesirable. One model for this is the electronic mail filtering language Sieve [4], which deliberately restricts itself from being Turing-complete.

Similar levels of safety and protection (though not automatic generation and parsing) could also be achieved through the use of a "sandbox" such as is used by Java applets, where strict bounds are imposed on the amount of memory, cpu time, stack space, etc., that a program can use. The difficulty with this approach is primarily in its lack of transparency and portability: unless the levels of these bounds are imposed by the standard, a bad idea so long as available resources are increasing exponentially with Moore's Law, a user can never be sure whether a particular program can successfully be executed on a given server without running into the server's resource limits, and a program which executes successfully on one server may fail unexpectedly on another. Non-expressively-complete languages, on the other hand, allow an implicit contract between the script writer and the server: so long as the script stays within the rules of the language, the server will guarantee that it will execute the script.

The second disadvantage with expressively complete languages is that they make automatic generation and parsing of scripts very difficult, as every parsing tool must be a full interpreter for the language. An analogy can be drawn from the document-creation world: while text markup languages like HTML or XML can be, and are, easily manipulated by smart editors, powerful document programming languages such as LaTeX or Postscript usually cannot be. While there are word processors that can save their documents in LaTeX form, they cannot accept as input arbitrary LaTeX documents, let alone preserve the structure of the original document in an edited form. By contrast, essentially any HTML editor can edit any HTML document from the web, and the high-quality ones preserve the structure of the original documents in the course of editing them.

11 Related work

11.1 IN service creation environments

The ITU's IN series describe, on an abstract level, service creation environments [6]. These describe services in a traditional circuit-switched telephone network as a series of decisions and actions arranged in a directed acyclic graph. Many vendors of IN services use modified and extended versions of this for their proprietary service creation environments.

11.2 SIP CGI

SIP CGI [9] is an interface for implementing services on SIP servers. Unlike a CPL, it is a very low-level interface, and would not be appropriate for services written by non-trusted users.

The paper "Programming Internet Telephony Services" [10] discusses the similarities and contrasts between SIP CGI and CPL in more detail.

12 Necessary language features

This section lists those properties of a call processing language which we believe to be necessary to have in order to implement the motivating examples, in line with the described architecture.

12.1 Language characteristics

These are some abstract attributes which any proposed call processing language should possess.

- Light-weight, efficient, easy to implement

  In addition to the general reasons why this is desirable, a network server might conceivably handle very large call volumes, and we don't want CPL execution to be a major bottleneck. One way to achieve this might be to compile scripts before execution.

- Easily verifiable for correctness

  For a script which runs in a server, mis-configurations can result in a user becoming unreachable, making it difficult to indicate run-time errors to a user (though a second-channel error reporting mechanism such as e-mail could ameliorate this). Thus, it should be possible to verify, when the script

is committed to the server, that it is at least syntactically correct, does not have any obvious loops or other failure modes, and does not use too many server resources.

- Executable in a safe manner

  No action the CPL script takes should be able to subvert anything about the server which the user shouldn't have access to, or affect the state of other users without permission. Additionally, since CPL scripts will typically run on a server on which users cannot normally run code, either the language or its execution environment must be designed so that scripts cannot use unlimited amounts of network resources, server CPU time, storage, or memory.

- Easily writeable and parsable by both humans and machines.

  For maximum flexibility, we want to allow humans to write their own scripts, or to use and customize script libraries provided by others. However, most users will want to have a more intuitive user-interface for the same functionality, and so will have a program which creates scripts for them. Both cases should be easy; in particular, it should be easy for script editors to read human-generated scripts, and vice-versa.

- Extensible

  It should be possible to add additional features to a language in a way that existing scripts continue to work, and existing servers can easily recognize features they don't understand and safely inform the user of this fact.

- Independent of underlying signalling details

  The same scripts should be usable whether the underlying protocol is SIP, H.323, a traditional telephone network, or any other means of setting up calls. It should also be agnostic to address formats. (We use SIP terminology in our descriptions of requirements, but this should map fairly easily to other systems.) It may also be useful to have the language extend to processing of other sorts of communication, such as e-mail or fax.

12.2 Base features -- call signalling

To be useful, a call processing language obviously should be able to react to and initiate call signalling events.

- Should execute actions when a call request arrives

  See section 7, particularly 7.1.

- Should be able to make decisions based on event properties

  A number of properties of a call event are relevant for a script's decision process. These include, roughly in order of importance:

  - Destination address

    We want to be able to do destination-based routing or screening. Note that in SIP we want to be able to filter on either or both of the addresses in the To header and the Request-URI.

  - Originator address

    Similarly, we want to be able to do originator-based screening or routing.

  - Caller Preferences

    In SIP, a caller can express preferences about the type of device to be reached -- see [11]. The script should be able to make decisions based on this information.

  - Information about caller or call

    SIP has textual fields such as Subject, Organization, Priority, etc., and a display name for addresses; users can also add non-standard additional headers. H.323 has a single Display field. The script should be able to make decisions based on these parameters.

  - Media description

    Call invitations specify the types of media that will flow, their bandwidth usage, their network destination addresses, etc. The script should be able to make decisions based on these media characteristics.

  - Authentication/encryption status

    Call invitations can be authenticated. Many properties of the authentication are relevant: the method of authentication/encryption, who performed the authentication, which specific fields were encrypted, etc. The script should be able to make decisions based on these security parameters.

- Should be able to take action based on a call invitation

  There are a number of actions we can take in response to an

incoming call setup request. We can:

- reject it

  We should be able to indicate that the call is not acceptable or not able to be completed. We should also be able to send more specific rejection codes (including, for SIP, the associated textual string, warning codes, or message payload).

- redirect it

  We should be able to tell the call initiator sender to try a different location.

- proxy it

  We should be able to send the call invitation on to another location, or to several other locations ("forking" the invitation), and await the responses. It should also be possible to specify a timeout value after which we give up on receiving any definitive responses.

o Should be able to take action based a response to a proxied or forked call invitation

  Once we have proxied an invitation, we need to be able to make decisions based on the responses we receive to that invitation (or the lack thereof). We should be able to:

- consider its message fields

  We should be able to consider the same fields of a response as we consider in the initial invitation.

- relay it on to the call originator

  If the response is satisfactory, it should be returned to the sender.

- for a fork, choose one of several responses to relay back

  If we forked an invitation, we obviously expect to receive several responses. There are several issues here -- choosing among the responses, and how long to wait if we've received responses from some but not all destinations.

- initiate other actions

  If we didn't get a response, or any we liked, we should be able to try something else instead (e.g., call forward on

busy).

12.3 Base features -- non-signalling

A number of other features that a call processing language should have do not refer to call signalling per se; however, they are still extremely desirable to implement many useful features.

The servers which provide these features might reside in other Internet devices, or might be local to the server (or other possibilities). The language should be independent of the location of these servers, at least at a high level.

- o Logging

  In addition to the CPL server's natural logging of events, the user will also want to be able to log arbitrary other items. The actual storage for this logging information might live either locally or remotely.

- o Error reporting

  If an unexpected error occurs, the script should be able to report the error to the script's owner. This may use the same mechanism as the script server uses to report language errors to the user (see section 12.5).

- o Access to user-location info

  Proxies will often collect information on users' current location, either through SIP REGISTER messages, the H.323 RRQ family of RAS messages, or some other mechanism (see section

6.2). The CPL should be able to refer to this information so a call can be forwarded to the registered locations or some subset of them.

- o Database access

  Much information for CPL control might be stored in external databases, for example a wide-area address database, or authorization information, for a CPL under administrative control. The language could specify some specific database access protocols (such as SQL or LDAP), or could be more generic.

- o Other external information

  Other external information a script could access includes web pages, which could be sent back in a SIP message body; or a clean interface to remote procedure calls such as Corba, RMI, or DCOM, for instance to access an external billing database. However, for simplicity, these interfaces may not be in the

initial version of the protocol.

12.4 Language features

Some features do not involve any operations external to the CPL's execution environment, but are still necessary to allow some standard services to be implemented. (This list is not exhaustive.)

- Pattern-matching

  It should be possible to give special treatment to addresses and other text strings based not only on the full string but also on more general or complex sub-patterns of them.

- Address filtering

  Once a set of addresses has been retrieved through one of the methods in section 12.3, the user needs to be able to choose a sub-set of them, based on their address components or other parameters.

- Randomization

  Some forms of call distribution are randomized as to where they actually end up.

- Date/time information

  Users may wish to condition some services (e.g., call forwarding, call distribution) on the current time of day, day of the week, etc.

12.5 Control

As described in section 8, we must have a mechanism to send and retrieve CPL scripts, and associated data, to and from a signalling server. This method should support reporting upload-time errors to users; we also need some mechanism to report errors to users at script execution time. Authentication is vital, and encryption is very useful. The specification of this mechanism can be (and probably ought to be) a separate specification from that of the call processing language itself.

13 Security Considerations

The security considerations of transferring CPL scripts are discussed in sections 8 and 12.5. Some considerations about the execution of the language are discussed in section 12.1.

14 Acknowledgments

We would like to thank Tom La Porta and Jonathan Rosenberg for their comments and suggestions.

15 Authors' Addresses

Jonathan Lennox
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA

EMail: lennox@cs.columbia.edu

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA

EMail: schulzrinne@cs.columbia.edu

16 Bibliography

[1] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[2] International Telecommunication Union, "Packet based multimedia communication systems," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[3] K. Coar and D. Robinson, "The WWW common gateway interface version 1.1", Work in Progress.

[4] T. Showalter, "Sieve: A mail filtering language", Work in Progress.

[5] J. Lennox and H. Schulzrinne, "CPL: a language for user control of internet telephony services", Work in Progress.

[6] International Telecommunication Union, "General recommendations on telephone switching and signaling -- intelligent network: Introduction to intelligent network capability set 1," Recommendation Q.1211, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1993.

[7] Guttman, E., Perkins, C., Veizades, J. and M. Day, "Service Location Protocol, Version 2", RFC 2608, June 1999.

[8] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K.

Schure, and H. Velthuijsen, "A feature interaction benchmark for IN and beyond," Feature Interactions in Telecommunications Systems, IOS Press, pp. 1-23, 1994.

[9] J. Lennox, J. Rosenberg, and H. Schulzrinne, "Common gateway interface for SIP", Work in Progress.

[10] J. Rosenberg, J. Lennox, and H. Schulzrinne, "Programming internet telephony services," Technical Report CUCS-010-99, Columbia University, New York, New York, Mar. 1999.

[11] H. Schulzrinne and J. Rosenberg, "SIP caller preferences and callee capabilities", Work in Progress.

17 Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

APPENDIX D

U.S. Patent No. 6,476,833 ("*Moshfeghi.*")

US0064768[illegible] 1

(12) **United States Patent**
Moshfeghi

(10) **Patent No.: US 6,476,833 B1**
(45) **Date of Patent: Nov. 5, 2002**

(54) **METHOD AND APPARATUS FOR CONTROLLING BROWSER FUNCTIONALITY IN THE CONTEXT OF AN APPLICATION**

(75) Inventor: **Mehran Moshfeghi**, Sunnyvale, CA (US)

(73) Assignee: **Koninklijke Philips Electronics N.V.**, Eindhoven (NL)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/281,393**

(22) Filed: **Mar. 30, 1999**

(51) **Int. Cl.**[7] **G06F 3/74**
(52) **U.S. Cl.** **345/854**; 345/762
(58) **Field of Search** 345/745, 742, 345/751, 759, 839, 744, 968, 853, 854, 762

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,784,564 A		7/1998	Camaisa et al.	
5,796,395 A	*	8/1998	De Hond	345/744
6,177,936 B1	*	1/2001	Cragun	345/781
6,266,060 B1	*	7/2001	Roth	345/853

FOREIGN PATENT DOCUMENTS

WO	WO9715008	4/1997	G06F/11/00
WO	WO9740446	10/1997	G06F/11/30
WO	WO9832076	7/1998	G06F/13/38
WO	WO9837480	8/1998	G06F/1/00
WO	WO 9845793	10/1998	G06F/17/30
WO	WO9848546	10/1998	H04L/29/06

OTHER PUBLICATIONS

"Look Ahead Filtering on Internet Content" IBM Technical Disclosure Bulletin, US IBM Corp. New York, vol. 40, No. 12, Dec. 1, 1997, pp. 143.

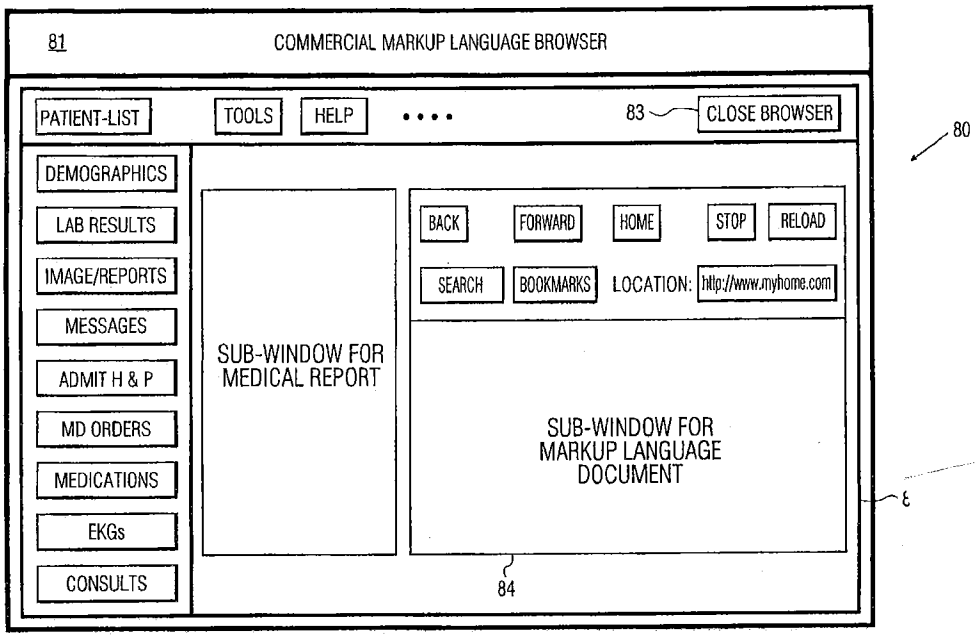
* cited by examiner

Primary Examiner—Cao H. Nguyen
(74) *Attorney, Agent, or Firm*—John Vodopia

(57) **ABSTRACT**

This invention includes methods and apparatus for browsing markup language documents from within the context of a client-server application running on an end-user device. Browser functionality, which is configured according to user profile information specifying each user's authorization and preferences, is embedded in the application, and can be activated by application controls. While some users have unrestricted authorization and access, others are restricted to certain browser functions and to certain allowed network resources. This restriction is enforced by preventing the browser functionality from generating network addresses that are not on a list of allowed network addresses also present in the user profile information. Network access restriction is achieved, in part, by filtering markup language documents before display to delete linking information that is not allowed. Document filtering methods are presented for Hypertext Markup Language (HTML) and extensible Markup Language (XML) documents. The document filtering methods are extendable to additional markup languages.

28 Claims, 8 Drawing Sheets



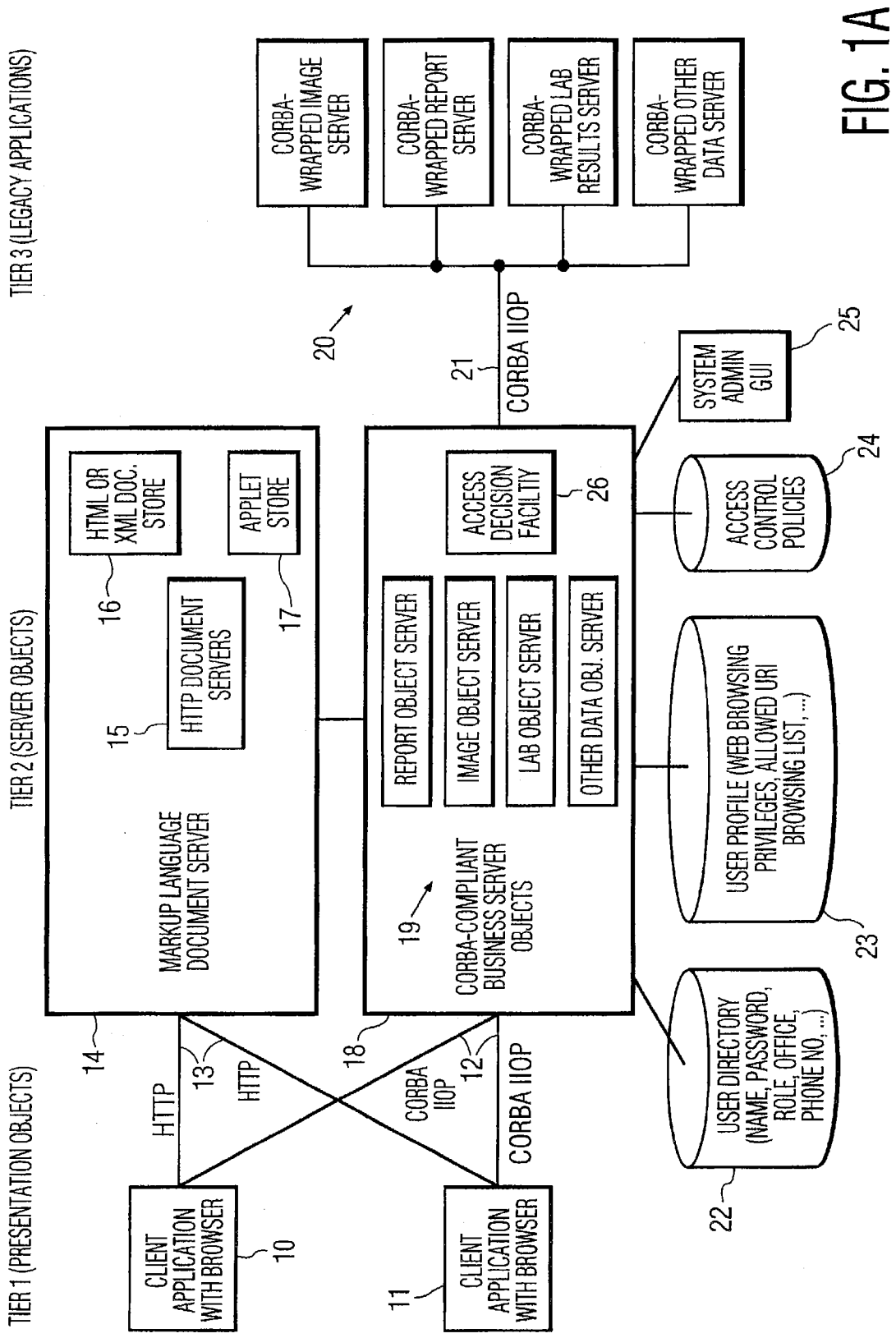

TIER 3 (LEGACY APPLICATIONS)
CORBA-WRAPPED IMAGE SERVER
CORBA-WRAPPED REPORT SERVER
CORBA-WRAPPED LAB RESULTS SERVER
CORBA-WRAPPED OTHER DATA SERVER
20
21
CORBA IIOP
SYSTEM ADMIN GUI
25
TIER 2 (SERVER OBJECTS)
HTML OR XML DOC. STORE
16
APPLET STORE
17
ACCESS DECISION FACILITY
26
ACCESS CONTROL POLICIES
24
15
HTTP DOCUMENT SERVERS
REPORT OBJECT SERVER
IMAGE OBJECT SERVER
LAB OBJECT SERVER
OTHER DATA OBJ. SERVER
MARKUP LANGUAGE DOCUMENT SERVER
19
CORBA-COMPLIANT BUSINESS SERVER OBJECTS
USER PROFILE (WEB BROWSING PRIVILEGES, ALLOWED URI BROWSING LIST, ...)
23
14
18
HTTP
13
HTTP
CORBA IIOP
12
CORBA IIOP
USER DIRECTORY (NAME, PASSWORD, ROLE, OFFICE, PHONE NO., ...)
22
TIER 1 (PRESENTATION OBJECTS)
CLIENT APPLICATION WITH BROWSER
10
11
CLIENT APPLICATION WITH BROWSER

FIG. 1A

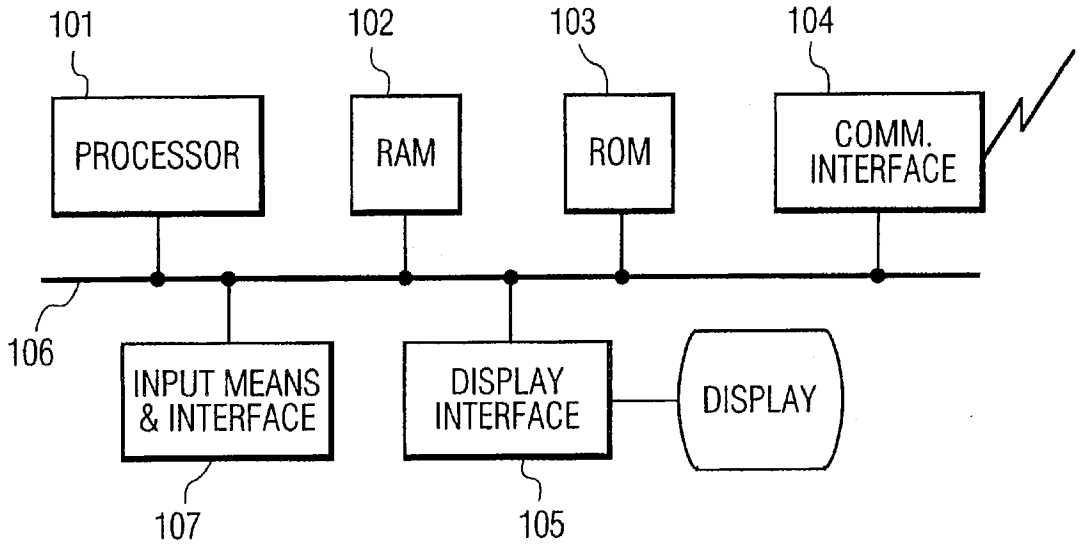

101
102
103
104
PROCESSOR
RAM
ROM
COMM. INTERFACE
106
INPUT MEANS & INTERFACE
DISPLAY INTERFACE
DISPLAY
107
105

FIG. 1B

A
B
C
D
E
F
G
H
I
J
K
L
M
POSSIBLE URIs
A = http://www.org1.com
B = http://www.org1.com/shopping/
C = http://www.org1.com/SCA/
D = http://www.org2.org/
E = http://www.org2.org/Member/
F = http://www.org2.org/TandS/
G = http://www.org2.org/TR/
H = http://www.org2.org/TR/WD-P3P-arch
I = http://www.org2.org/TR/WD-xlink
J = http://www.org3.com
K = http://www.org3.com/homebus/html
L = http://www.org3.com/homeins/html
M = http://www.org3.com/homehome/html
DISALLOWED URIs
REPRESENTATIVES OF ALLOWED URIs
http://www.org1.com/shopping/ B
http://www.org1.com/SCA/* C AND ALL CHILDREN
http://www.org2.org/ D
http://www.org2.org/TandS/* F AND ALL CHILDREN
http://www.org2.org/TR/ G
http://www.org2.org/TR/WD-xlink I
http://www.org3.com/* J AND ALL CHILDREN

FIG. 2

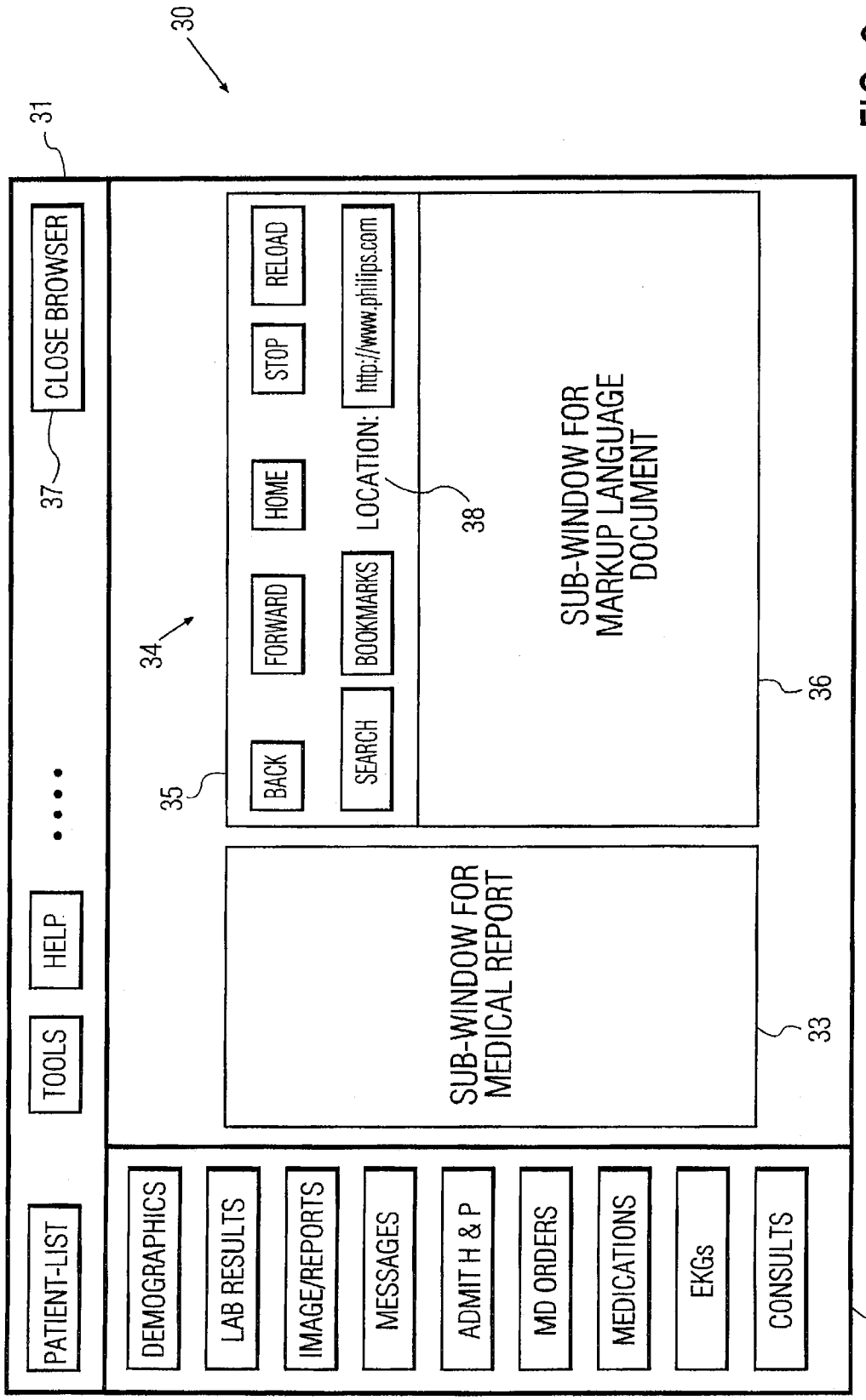

30
31
PATIENT-LIST
TOOLS
HELP
• • • •
CLOSE BROWSER
37
DEMOGRAPHICS
LAB RESULTS
IMAGE/REPORTS
MESSAGES
ADMIT H & P
MD ORDERS
MEDICATIONS
EKGs
CONSULTS
32
34
35
BACK
FORWARD
HOME
STOP
RELOAD
SEARCH
BOOKMARKS
LOCATION:
http://www.philips.com
38
SUB-WINDOW FOR MARKUP LANGUAGE DOCUMENT
36
SUB-WINDOW FOR MEDICAL REPORT
33

FIG. 3

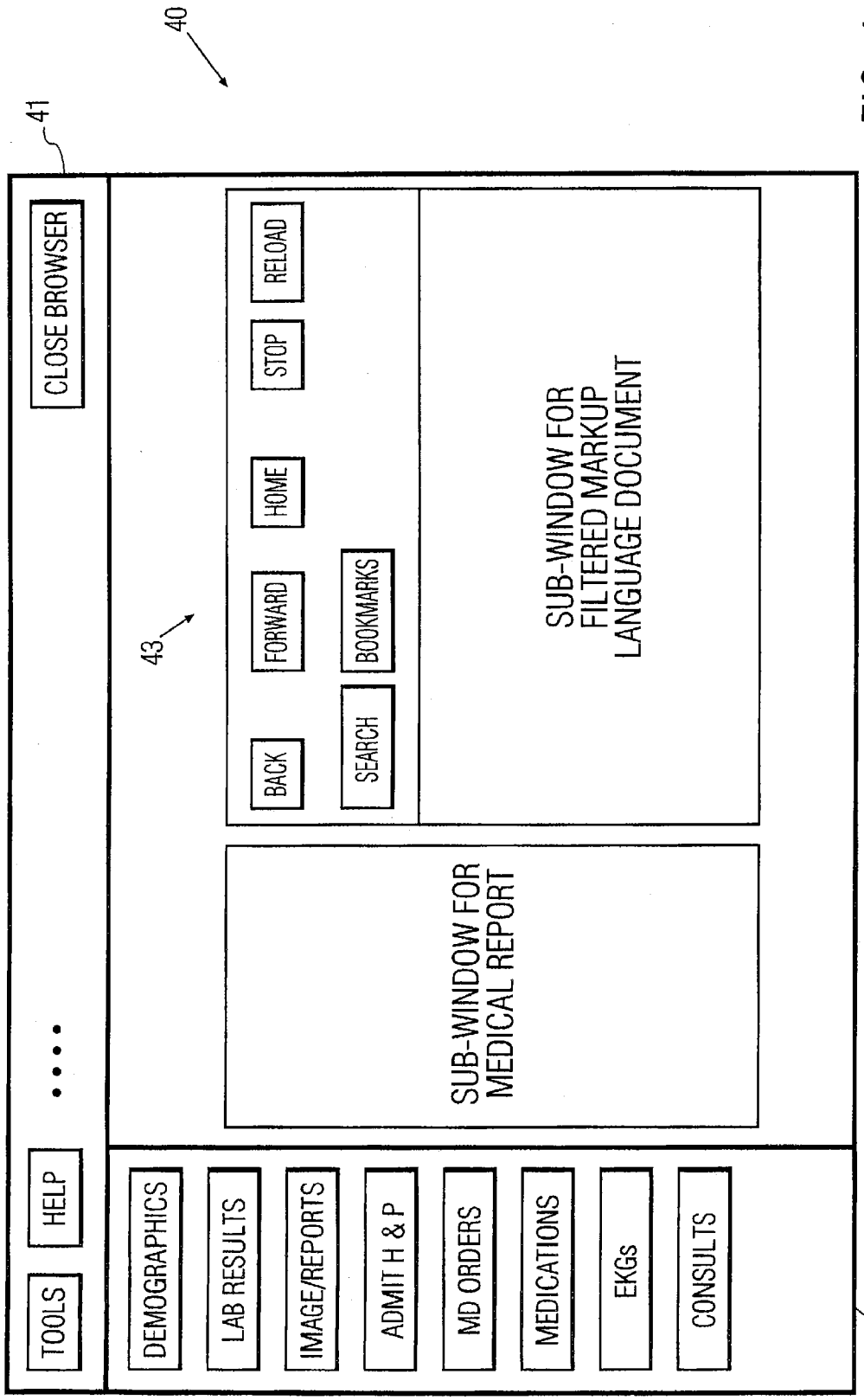

40
41
CLOSE BROWSER
TOOLS
HELP
• • • •
43
BACK
FORWARD
HOME
STOP
RELOAD
SEARCH
BOOKMARKS
SUB-WINDOW FOR FILTERED MARKUP LANGUAGE DOCUMENT
SUB-WINDOW FOR MEDICAL REPORT
DEMOGRAPHICS
LAB RESULTS
IMAGE/REPORTS
ADMIT H & P
MD ORDERS
MEDICATIONS
EKGs
CONSULTS
42

FIG. 4

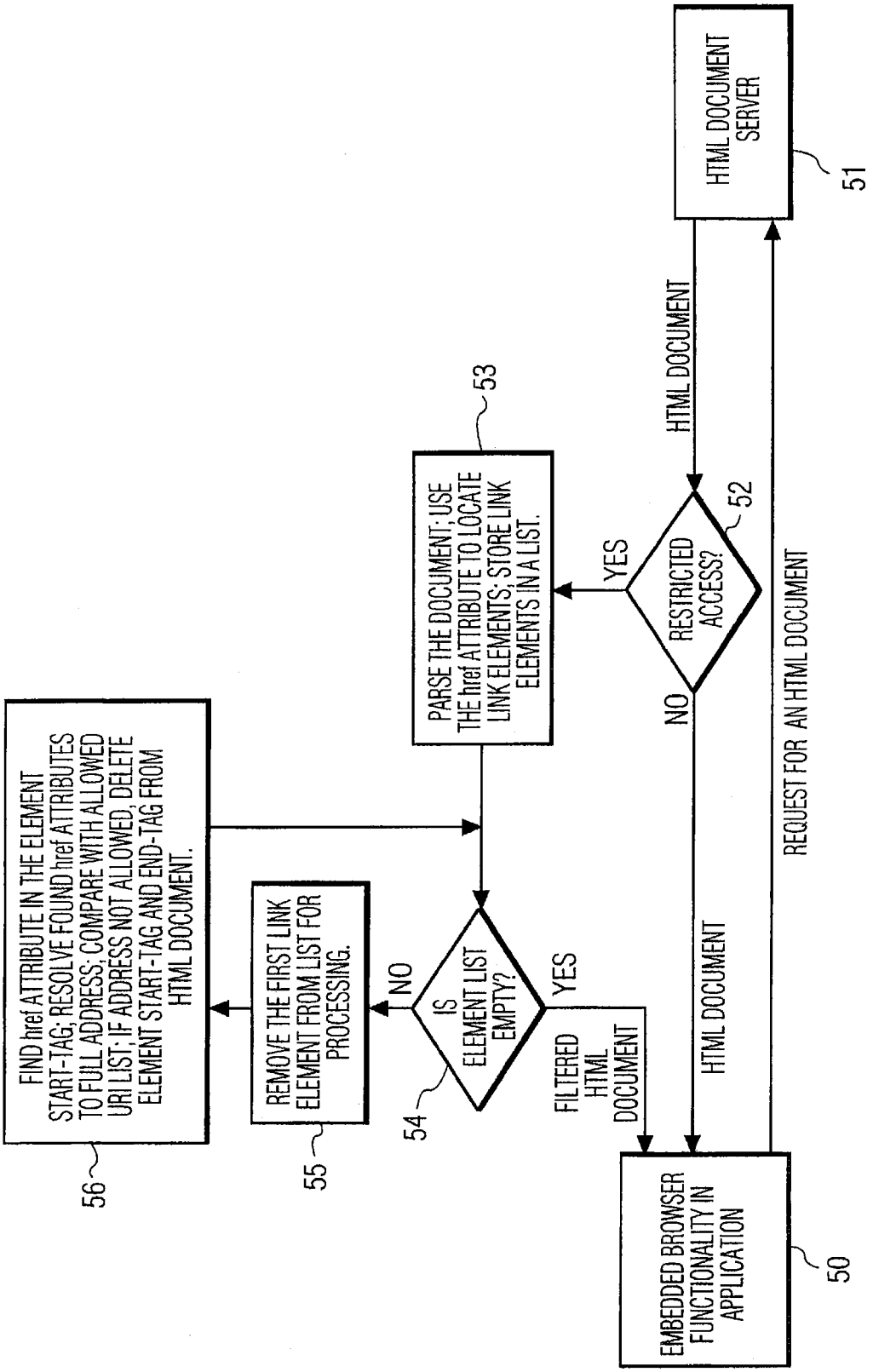

HTML DOCUMENT SERVER
51
HTML DOCUMENT
RESTRICTED ACCESS?
52
YES
NO
PARSE THE DOCUMENT; USE THE href ATTRIBUTE TO LOCATE LINK ELEMENTS; STORE LINK ELEMENTS IN A LIST.
53
IS ELEMENT LIST EMPTY?
54
NO
YES
REMOVE THE FIRST LINK ELEMENT FROM LIST FOR PROCESSING.
55
FIND href ATTRIBUTE IN THE ELEMENT START-TAG; RESOLVE FOUND href ATTRIBUTES TO FULL ADDRESS; COMPARE WITH ALLOWED URI LIST; IF ADDRESS NOT ALLOWED, DELETE ELEMENT START-TAG AND END-TAG FROM HTML DOCUMENT.
56
FILTERED HTML DOCUMENT
HTML DOCUMENT
REQUEST FOR AN HTML DOCUMENT
EMBEDDED BROWSER FUNCTIONALITY IN APPLICATION
50

FIG. 5

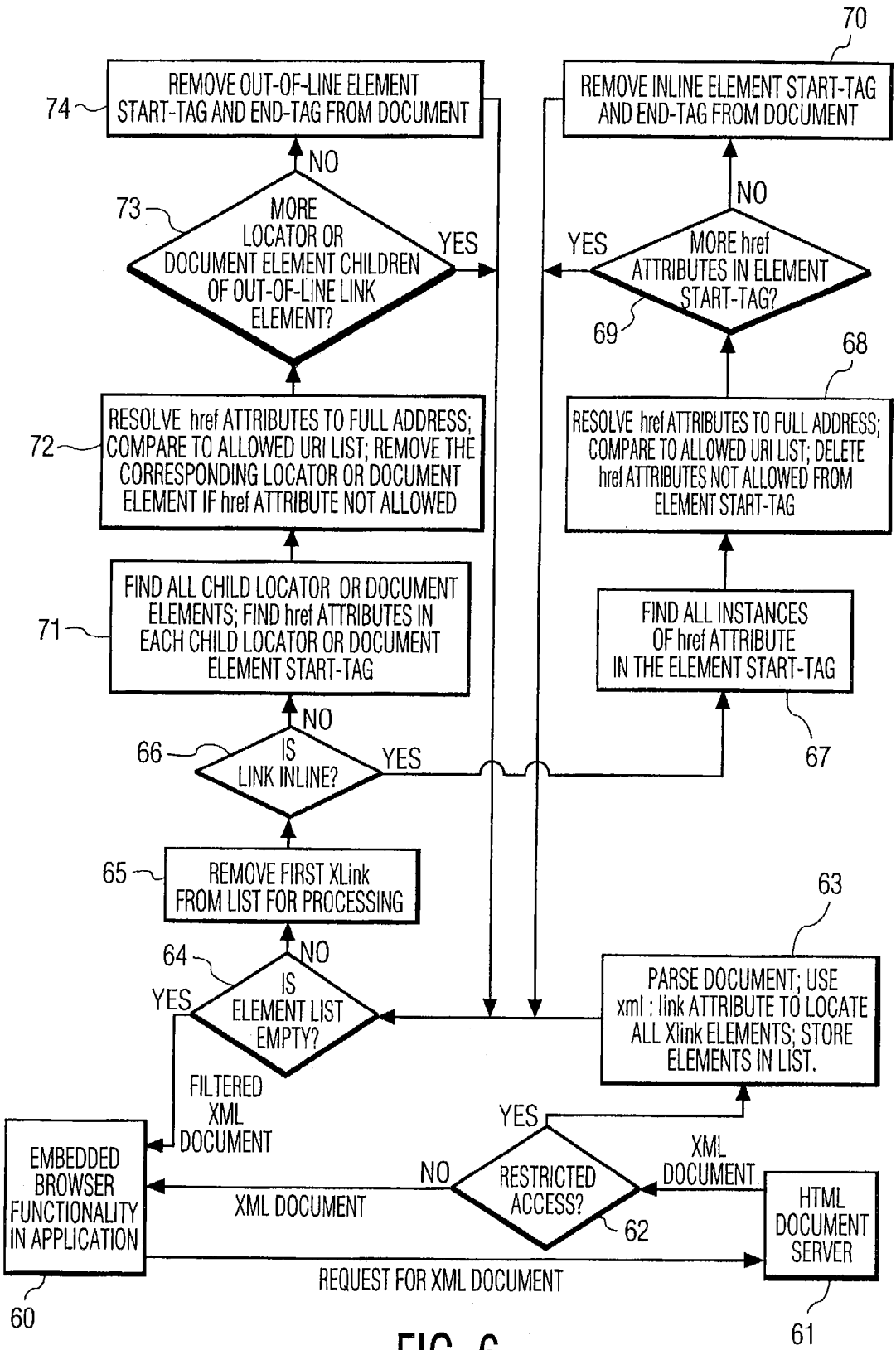

70
REMOVE INLINE ELEMENT START-TAG AND END-TAG FROM DOCUMENT
NO
YES
MORE href ATTRIBUTES IN ELEMENT START-TAG?
69
68
RESOLVE href ATTRIBUTES TO FULL ADDRESS; COMPARE TO ALLOWED URI LIST; DELETE href ATTRIBUTES NOT ALLOWED FROM ELEMENT START-TAG
FIND ALL INSTANCES OF href ATTRIBUTE IN THE ELEMENT START-TAG
67
74
REMOVE OUT-OF-LINE ELEMENT START-TAG AND END-TAG FROM DOCUMENT
NO
73
MORE LOCATOR OR DOCUMENT ELEMENT CHILDREN OF OUT-OF-LINE LINK ELEMENT?
YES
72
RESOLVE href ATTRIBUTES TO FULL ADDRESS; COMPARE TO ALLOWED URI LIST; REMOVE THE CORRESPONDING LOCATOR OR DOCUMENT ELEMENT IF href ATTRIBUTE NOT ALLOWED
71
FIND ALL CHILD LOCATOR OR DOCUMENT ELEMENTS; FIND href ATTRIBUTES IN EACH CHILD LOCATOR OR DOCUMENT ELEMENT START-TAG
NO
66
IS LINK INLINE?
YES
65
REMOVE FIRST XLink FROM LIST FOR PROCESSING
63
64
NO
IS ELEMENT LIST EMPTY?
YES
PARSE DOCUMENT; USE xml : link ATTRIBUTE TO LOCATE ALL Xlink ELEMENTS; STORE ELEMENTS IN LIST.
FILTERED XML DOCUMENT
YES
XML DOCUMENT
NO
RESTRICTED ACCESS?
62
EMBEDDED BROWSER FUNCTIONALITY IN APPLICATION
XML DOCUMENT
HTML DOCUMENT SERVER
REQUEST FOR XML DOCUMENT
60
61

FIG. 6

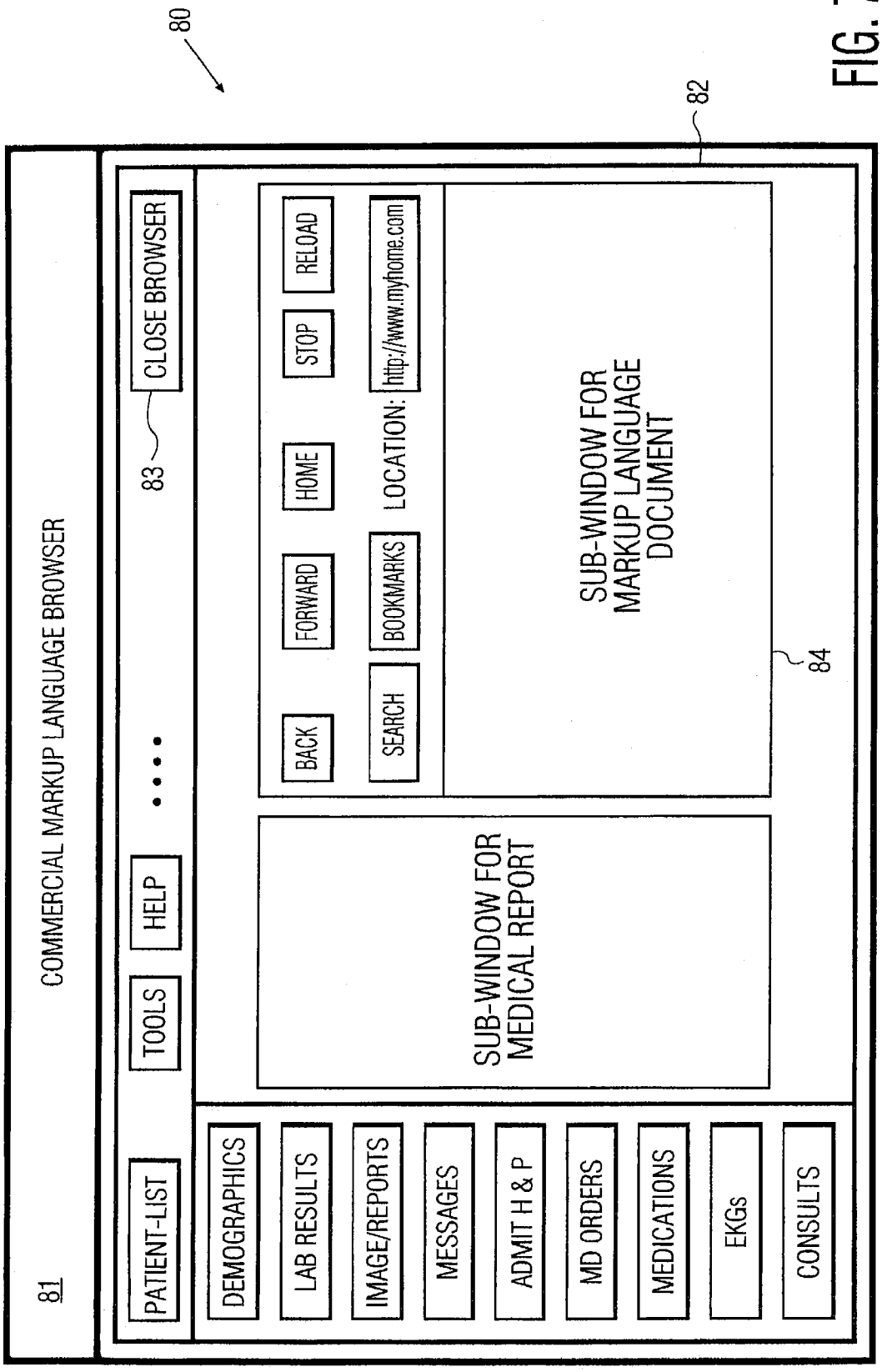

80
81
COMMERCIAL MARKUP LANGUAGE BROWSER
PATIENT-LIST
TOOLS
HELP
• • • •
83
CLOSE BROWSER
82
BACK
FORWARD
HOME
STOP
RELOAD
SEARCH
BOOKMARKS
LOCATION: http://www.myhome.com
SUB-WINDOW FOR MARKUP LANGUAGE DOCUMENT
84
SUB-WINDOW FOR MEDICAL REPORT
DEMOGRAPHICS
LAB RESULTS
IMAGE/REPORTS
MESSAGES
ADMIT H & P
MD ORDERS
MEDICATIONS
EKGs
CONSULTS

FIG. 7

METHOD AND APPARATUS FOR CONTROLLING BROWSER FUNCTIONALITY IN THE CONTEXT OF AN APPLICATION

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention includes methods and apparatus relating generally to browsing markup language documents across a network from within the context of a client application program, and specifically to restricting access only to allowed network resources and to allowed browser interface functions.

2. Description of Related Art

The public Internet and private intranets allow client programs on one computer to exchange information with server programs on other computers. World Wide Web (WWW) browsers are a type of client program that provide easy-to-use, virtually standardized, graphical access for browsing hypertext information formatted in markup language documents.

However, most business applications perform other non-browsing tasks and require specialized interfaces. Attempts to provide a browser-like interface for application-oriented business tasks are likely to lead to undesirable user interfaces. See, e.g., Nielson, 1998, "Does Internet=Web?", Jakob Nielson's Alertbox for Sep. 20, 1998. For example, e-mail applications like Lotus Notes, Outlook Express or Eudora are more professional with greater functionality than simple browser-like WWW e-mail interfaces (e.g., Hotmail). Likewise, other vertical market stand-alone business applications provide more features and functions than can be supported in a standard browser.

However, it is often useful to provide for markup language browsing while remaining in the application context. In one alternative, the application may have a sub-window with some of the functionality of a standard browser. This window is not a separate window but is an embedded window that is an integral part of the application. Such a feature allows browsing of hypertext information without needing to exit the application and starting a separate browser application, such as Navigator of Netscape Corp. or Internet Explorer of Microsoft.

Accessing and displaying markup language documents in the application context is typically different from standard browsing sessions. In standard browsing, a user can enter any Uniform Resource Identifier (URI) or click on any displayed link to access any desired content or resource. See, e.g., Internet Engineering Task Force, 1994, Uniform Resource Locators (URLs) RFC 1738; Internet Engineering Task Force, 1995, Relative Uniform Resource Locators—RFC 1808. However, in order to remain productive, browsing from the application context, should typically be focused and related to the application. For example, it can be more productive to permit access only to relevant documents and resources.

Accordingly, it is important that any browser functionality embedded in the application context be configurable for focused productive use.

Markup languages are widely used, especially for Internet content. Generally, markup languages are documents with text, images, and embedded markup tags which specify the desired display formatting of the document. In the past, Internet markup language hypertext documents have been largely limited to the Hypertext Markup Language (HTML).

However, new markup languages are being introduced for specialized and general use. For example, special markup languages are being developed which are designed for use in consumer appliances. A newer general markup language is the extensible Markup Language (XML), being developed by the World Wide Web Consortium (W3C). XML may become a virtually universal markup language of the Internet in the future. Most producers of commercial browser have committed to supporting XML in future releases.

In detail, XML is a restricted form of the Standard Generalized Markup Language (SGML), which is also designed for use on the WWW. XML has many potential applications such as separation of structure from presentation, intelligent searching, messaging formats, and data converters.

XML became a W3C recommendation in February 1998. See, e.g., World Wide Web Consortium, 1998, extensible Markup Language (XML) Version 1.0, Recommendation 10. The W3C is developing a number of XML-related standards. An important standard is the XML Linking Language, or XLink. See, e.g., World Wide Web Consortium, 1998, XML Linking Language (XLink), Working Draft 3-March. XLink describes linking information constructs that can be inserted into XML resources or documents to describe links between resources. XLink uses XML syntax to create one directional links (like HTML) as well as more complex multidirectional links.

Other XML standards include the extensible Stylesheet Language (XSL) being designed for presentation of XML documents. See, e.g., World Wide Web Consortium, 1998, extensible Stylesheet Language (XSL), Version 1.0. XML name spaces are being designed to provide a simple method for qualifying element and attribute names used in XML documents by associating them with name spaces identified by URI references. See, e.g., World Wide Web Consortium, 1998, Name spaces in XML. The Document Object Model (DOM) is designed for modeling document elements as objects and for providing an interface allowing programs and scripts to dynamically access and update document content, structure, and style. See, e.g., World Wide Web Consortium, 1998, Document Object Model (DOM) Level 1 Specification, Version 1.0. The XML Pointer Language (XPointer) is designed for reference to elements, character strings, and other parts of XML documents. See, e.g., World Wide Web Consortium, 1998, XML Pointer Language (Xpointer).

Accordingly, it is also important that browser functionality embedded in the application context be adaptable to HTML, XML and other markup languages being developed.

Citation of a reference herein, or throughout this specification, is not to be construed as an admission that such reference is prior art to the Applicant's invention of the matter subsequently claimed.

SUMMARY OF THE INVENTION

General objects of the present invention are to provide methods and apparatus which overcome the above identified requirement and lacks in the current art.

Specific objects of the present invention include methods and apparatus providing configurable markup language browser functionality embedded in the context of a client-server application running on standard end-user devices. The browser functionality of this invention is configurable so that the network resources accessible by a user are restricted in order to focus the user's attention to determined relevant content. Such configuration may als

used to block possibly offensive documents. Further, the browser functionality is configurable so that certain users are prevented for utilizing certain browser features. The browser configuration and allowed network access are separately customizable for each user. This invention is applicable both to HTML and XML documents and also to documents formatted in emerging markup languages.

Accordingly, this invention includes methods and apparatus for browsing markup language documents from within the context of an executing application. This is achieved by embedded browser functionality that can be activated to display an embedded browser sub-window inside the main application window. While the prior art provides the same resource access rights and browser interface functions to all users, the present invention provides methods and apparatus where some users are allowed to access any resource on the public Internet or private intranets, while other users can only access the limited lists of resources set forth in the content of their user profile. Further, the functionality and appearance of the user interface of the embedded browser sub-window is also configured from information in the user profile. The administrator of the system in which an application according to this invention is installed can change the resource access and browsing function privileges of users by editing the content of their user profiles.

Network resource access is restricted by preventing the embedded browser functionality from generating linking information that is not allowed. In a preferred embodiment, a user with restricted Web browsing privileges is not presented any location field in the browser sub-window, into which URIs that are not allowed can be entered by the user. Also, in the preferred embodiment, if an allowed document is delivered from its server with links to other documents that are not allowed, these links are filtered from the document before it is displayed in the browser sub-window. Therefore, a restricted user can not access documents that are not allowed by activating displayed linking information. Such a user can access network resources only by allowed links present in displayed and filtered markup language documents. Optionally, the browser dynamically creates a personalized user home page listing all links allowed to the user.

Filtering methods are described for HTML and for XML, These methods can also be applied to other emerging markup languages, such as those being developed for consumer appliances. Preferably, the application is implemented in Java. If so, optionally, it can be an applet downloaded and executed in commercially available markup language browsers.

In a first embodiment this invention includes an apparatus for a user to browse markup language documents, the documents being stored for retrieval at one or more computer systems attached to a network and containing linking information representing resource addresses, the apparatus comprising: an end-user device comprising processor, memory, network attachment, and display, wherein the memory stores user profile records whose contents describe the characteristics and preferences of the user, an application program comprising (i) instructions for causing the processor to perform an application function which displays an application window including application specific controls and data, wherein the application specific controls are responsive to the content of the user profile records and comprise a browser control, and (ii) instructions for causing the processor to perform, responsive to user activation of the browser control, a browser function which displays within the application window a browser sub-window including browser specific controls and markup language documents, wherein both the displayed browser specific controls and any linking information retained present in the displayed markup language documents are responsive to the content of the user profile records.

In a first aspect of the first embodiment, this invention includes the apparatus of the first embodiment wherein the browser specific controls comprise a location entry field for entry of linking information representing resource addresses, and wherein display or not of the location entry field is responsive to the contents of the user profile records.

In a second embodiment, this invention includes a method for a user to browse markup language documents at an end-user device, the documents being stored for retrieval at one or more computer systems attached to a network and containing linking information representing resource addresses, the method comprising: entering user identification information, authenticating entered user identification information and loading user profile records associated with the authenticated user to the end-user device, displaying an application window at the end-user device including application specific controls and data, wherein the application specific controls are responsive to the content of the user profile records and comprise a browser control, displaying, within the application window and responsively to user activation of the browser control, a browser sub-window including browser specific controls and markup language documents, wherein both the displayed browser specific controls and any linking information retained in the displayed markup language documents are responsive to the content of the user profile records.

BRIEF DESCRIPTION OF THE DRAWING

Other objects, features and advantages of the present invention will become apparent upon perusal of the following detailed description when taken in conjunction with the appended drawing wherein:

FIGS. **1**A–B illustrates an exemplary client-server architecture and end-user device to which this invention is applicable;

FIG. **2** illustrates an exemplary forest of representative tree-structured URIs;

FIG. **3** illustrates a graphical interface of an application with embedded browser functionality presented to a user of a certain higher authority according to this invention;

FIG. **4** illustrates a graphical interface of an application with embedded browser functionality presented to a user of a different lower authority according to this invention;

FIG. **5** illustrates a method for filtering an HTML document by deleting linking information that is not allowed;

FIG. **6** illustrates a method for filtering an XML document by deleting linking information that is not allowed; and

FIG. **7** illustrates a graphical interface of a signed applet or a policy-based authorized applet with an embedded browser running inside a commercial browser program.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the following, in section 1, an exemplary client-server system architecture, to which the methods and apparatus of the present invention are advantageously applied, is first described. Next, in section 2, detailed descriptions of the browser functionality of the present invention and its implementation are presented. Finally, in section 3, methods for filtering markup language documents are described Where

necessary, the detailed description also includes concise discussions of necessary technical background.

1. Client-server System

1.1. Architecture Overview

FIG. **1A** illustrates an exemplary 3-tier client-server architecture applied to computerized medical records (CPR) distribution. This architecture is based jointly on a distributed object architecture in which Common Object Request Broker Architecture (CORBA) compliant object request brokers (ORB) communicate using Internet Inter-ORB Protocol (IIOP) to connect and on markup language document transfer from servers to clients using the Hypertext Transfer Protocol (HTTP). The network illustrates is preferably the public Internet or private intranets. The CORBA family of distributed object standards are developed and published by the Object Management Group. The HTTP and related internet protocols are developed and published by the Internet Engineering Task Force.

This client-server system of this figure, which is described in this subsection, is exemplary, being used for concreteness and ease of the subsequent description. The system and architecture is not to be understood as limiting, since the methods and apparatus of this invention are equally applicable to other client-server architectures, such as 2-tier client-server architectures, as well as to client-server architectures not based on CORBA. For example, if the illustrated business server objects and the front-end clients are all implemented in Java then Sun Microsystems' Remote Method Invocation (RMI) technology can be used instead of the CORBA architecture. Alternatively the Distributed Component Object Model (DCOM) of the Microsoft Corp. can be used in place of CORBA.

Further, this exemplary architecture is applied, again for purposes of concreteness and ease of description only, to the distribution of computerized medical records for physicians, patients, and other users. This application is also to be understood as not being limiting, since one of ordinary skill in the art will appreciate how to adapt this invention to other application fields. Accordingly, the present invention encompasses not only application to CPR distribution, but to other vertical market domains and their applications, such as banking, insurance, airlines, hotels, transportation, and so forth.

With reference again to FIG. **1A**, the first tier of the three tier system is illustrated to comprise end-user devices **10** and **11**, which in turn comprise, at least, a processor, static and dynamic memory, a display, input means, and a communication interface. The end-user devices can be standard PCS, such as desktop, workstation or notebook machines. They can equally be Internet appliances, such as WebTVs, screen-telephones, PDAs, etc., or even other types of information appliances with communication capabilities.

The end-user devices run front-end client software including an application providing visual display of business data, markup language documents, and other resources available across the public Internet or private intranets. A user interacts with the user interface on the display of the end-user device presented by the client application software to send or request information from network-connected, tier-2 server systems.

Preferably, the application software is structured in a distributed object-oriented manner, so that, for example, the business data appears as business server objects **19**. Also it is preferable that distributed objects be structured as CORBA-compliant client objects that interact with distributed CORBA-compliant server objects. CORBA clients and servers, using ORBs resident on their end-user devices or computers, communicate using IIOP over exemplary network links **12**. Client objects include stubs generated using CORBA's Interface Definition Language (IDL) in order to remotely invoke methods on the distributed server objects. Finally, this software is also preferably implemented in the Java language in a platform-independent manner and installable as either a complete application or as an applet downloadable from a markup language document server for use in a browser.

According to this invention, the application software also includes markup language document browser functionality. Such browser functionality preferably acts as a client using the HTTP protocol to request markup language documents from servers over exemplary network links **13**. The browser functionality then displays the received documents as instructed by the formatting markups in the documents. Secure HTTP (s-HTTP) is preferably used where secure HTTP connections are desired. The secure socket layer (SSL) protocol can also provide security at the socket layer for IIOP communication. The system illustrated in FIG. **1A** allows both browser/HTTP and ORB/IIOP requests and responses to flow on the same network.

The middle tier, or tier **2**, of the three tier client-server architecture is illustrated to include markup language document server systems **14** and business server object systems **18**. Document server systems **14** are illustrated as running HTTP document server function **15**, which responds to requests from HTTP clients, and provides, among other resources, HTML, XML, or other markup language documents from document store **16**, and Java applets from applet store **17**. Applets according to the HTML and XML markup languages are referenced with the <APPLET> tag. One or more HTTP document server functions and associated stores can be resident on one or more HTTP server systems, and can be physically located anywhere on the network.

Business server systems **18** are illustrated as running CORBA-compliant business-server objects **19**, which individually respond to remote method invocations for specific type of data from client objects, which in turn are part of the application software running in the end-user devices. The CORBA-compliant business-server objects implement basic business logic of the business application of the client-server architecture, and thereby provide a dynamically generated and integrated view of the different tier-3 data servers. Illustrated in FIG. **1A** are separate business server objects for particular classes of data stored in the client-server architecture; alternately, one business-server object can respond to requests for more than one type of data. The business server objects can run in the same system process, in different system processes, on the same computer system, or on different computer systems. Although illustrated as resident on separate system **14** and **18**, alternatively HTTP servers and CORBA servers can also be co-resident on the same computer system.

The third tier, or tier **3**, is illustrated to include one or more data-server systems **20** communicating with business-server object systems **18** over exemplary network links **21**. These data-server systems run data-server objects that present a CORBA-compliant object-oriented interface for the various classes of data stored on these systems. Data server objects can include existing non-object-oriented legacy data stores and applications that have been interfaced using CORBA-compliant IDL, i.e., "CORBA-wrapped" legacy systems. See, e.g., Orfali et al., 1997, Client/Server Programming with Java and CORBA, ISBN 0-471-16351-1, John Wiley & Sons, 1997 (a CORBA and Java reference). Alternatively, these data-server objects can be initially

designed in an object-oriented fashion with CORBA-compliant interfaces. Such systems do not need to be CORBA-wrapped.

For concreteness, FIG. 1A illustrates an exemplary application of this three-tier client-server architecture to computerized patient record (CPR) distribution. Accordingly, second tier business-server objects and corresponding third tier data-server objects are illustrated for data types such as medical reports, medical images, lab results, and other types of medical data. Other appropriate business server objects not illustrated can provide general system functions such as session management, medical event notification, system security, display personalization, paging notification, patient-list retrieval, bookmarking, patient arrival and discharge, patient demographics, message exchange, database connection management, and so forth.

1.2. The End-user Device

The application software with the embedded browser functionality according to this invention runs on programmable end-user devices with network communication capabilities. These can range from specialized information appliances to standard PCs, as known in the art.

As illustrated in FIG. 1B, these devices preferably include, at least, the following common structures known in the art: microprocessor **101**, long-term ROM memory **103**, dynamic RAM memory **102**, display interface and display **105**, communication interface **104**, input interface and means **107**, and interconnect **106**. Long-term memory can also include magnetic and optical devices. The display can be, for example, an LCD with an LCD controller. The input means can be such known devices as a keyboard, mouse, touch screen, buttons, voice recognition, and so forth, together with needed interface elements. These interconnect can be an external bus, or alternatively, when one or more of the illustrated structures are implemented together on a single VLSI chip, the interconnect can be a specialized on-chip interconnect along with an optional external bus.

1.3. System Administration Databases

The client application software of this invention, and especially its embedded browser functionality, which runs in the end-user devices, is configured according to data descriptive of the users of the system, in particular of their authorizations and access rights. In a preferred embodiment, this user-descriptive data is stored in data stores associated with the tier-2 server systems, in particular in user directory database **22** and user profile database **23**. Although these databases are illustrated in FIG. 1A as being connected to object server system **18**, they are preferably connected to whatever tier-2 server systems has need of the stored data. These data stores are described in this subsection.

User directory **22** preferably stores for each user authentication information, role information, access control information, general information, and so forth. Authentication information includes data used to identify a user as authorized to access the client-server system at all. This data can include user name and password, or biometric data, such as fingerprint, or so forth. Role information includes data indicative of a user's role in the institution using the client-server system. This data can indicate, for example, that a user is a physician along with speciality and seniority, or a nurse along with speciality and seniority, or a patient, or a legal guardian of a patient, and so forth. Access control information includes data indicative of a user's authority to access and update the various types of data stored through the system. For example, a system administrator is authorized to read and update the user directory and user profile databases. General user information can include such data as office or ward location, phone number, application personalization preferences, and so forth. The user directory can be implemented and accessed by commercially available directory software known in the art, such as that implementing the Lightweight Directory Access Protocol (LDAP). LDAP is a widely used protocol for accessing, for example, an X.500 directory server, a Novel directory server, a Netscape directory server, and so forth.

User profile database **23** stores information for configuring the embedded browser functionality present in the client application software of this invention. Browser functionality, especially the network access rights allowed to different users is typically decided by the administration of the client-server system in order to promote focused and productive use of the application. For example for a CPR application, physician users may be allowed unrestricted network access in order to search for medical information anywhere on the public Internet from within the context of the CPR client application. On the other hand, nurse users may be limited only to particular nursing related documents available on the group's or hospital's own intranet. Finally, patients may be even more limited to access only certain educational documents relating to their particular diagnoses. Code obfuscation or encryption techniques may be used to prevent tampering with downloaded user profile information.

Accordingly, FIG. 1A illustrates system administration graphical user interface (GUI) **25**, for example provided on an administration workstation, by which a system administrator, or other designated user, enters and updates the contents of the user profile database **23** in accord with the functionality allowed to the various users. The same system administrator would typically also similarly enter and update user directory **22**.

In a preferred embodiment, the user profile database includes at least the following types of information for each user. First, there is included an indication of whether the user has unrestricted or restricted network access from the embedded browser functionality. Second, if the user has restricted network access, then the user profile includes representations of all the linking information addressing of all the network resources allowed to the user. Preferably, the linking information is in the format of universal resource identifiers (URI), and the user profile then has representations of the URIs of all allowed markup language documents. Herein, "linking information" is used to identify network resource addressing information of all types and formats. Third, the user profile preferably also includes indications of which other configurable aspects of browser functionality, such as the browser controls displayed to the user, support for markup language document forms, tables, and frames, applets, document printing, cookies, multithreading, and so forth, are allowed to the user.

Although illustrated as two separate databases, in alternate implementations user directory database **22** and user profile database **23** may be stored together in a single database.

In addition to control of embedded browser functionality, a separate and independent control prevents unauthorized access of a user to the business-server objects. Requests from the client applications to the business-server objects are intercepted by access decision facility **26**, which checks whether the requesting user, who has the particular access control information stored in the user directory, is authorized to access the information requested in view of the access control policies stored in access control policy database **24**. If the access control policies grant access, the server is

allowed to respond with the requested information to the client application. Otherwise, access is denied, and, for example, the service denial is logged to a log file for security audit purposes.

1.4. Representation of Allowed URI List

In the preferred embodiment, the user profile records for a user a representation of the URIs of all allowed network resources, such as markup language documents. The allowed URIs are selected from the typically large number of all possible URIs that can be accessed by the user. The set of all possible URIs can generally be structured as a tree, or as a forest of trees, or, generally, as a directed graph of linked URIs. Therefore, the allowed URIs are a sub-tree, a sub-forest, or a sub-graph of all possible URIs. Any representations of sub-trees, sub-forests, or sub-graphs, as appropriate, that are known and apparent to one of ordinary skill in the arts can be advantageously applied to represent all allowed URIs in the user profile database. In this subsection, a preferred such representation of all allowed URIs is described with reference to FIG. **2**, which illustrates an exemplary forest of three trees of possible URIs.

In FIG. **2**, nodes A, D, and J represent root URIs for the homes pages of Organization-**1**, Organization-**2**, and Organization-**3**, respectively. Children of a parent node extend the parent URI by one additional relative address. For example, child node G extends the URI of parent node D with the additional relative address of "TR/". Likewise, child node I extends the URI of parent node G with the additional relative address of WD-xlink. The list of all possible URIs is illustrated on the left in FIG. **2**. Of these, this particular user is authorized to access all URIs except those for the nodes A, E, and H, which are indicated by arrow in the illustrated list and by arrowheads in the tree diagrams.

A preferred representation of the allowed URIs for storage in this user's profile is illustrated on the right in FIG. **2**. In this representation, all URIs are the respective absolute addresses of allowed nodes. A URI followed the character "*" indicates that both the node having this URI address and all its child nodes are allowed. Nodes C, F and J are examples of allowed nodes for which all child nodes are allowed, and which accordingly have a "*" after their address in the allowed list. This representation is considerably more compact than the alternative in which the URI of every allowed node is individually listed. Note that it is possible to limit the granularity of allowed content. For example, in FIG. **2**, although the home page for Organization-**1**, node A, is not allowed, children of the home page are allowed.

In an alternative representation, the URIs listed are the absolute addresses of nodes that are not allowed. Here also, a special character can be used to indicate that a node and all its child nodes are not allowed. This alternative representation is preferable in the case where a user is allowed to access most resources, only a few resources not being allowed. Such a case may occur, for example, if it desired to filter offensive or inappropriate information from appearing in the context of the client application program.

Both representations may be combined in a single user profile database by including an indication of which representation applies to a particular user or to a particular group of URIs in the list for a particular user. The subsequently described markup document filtering methods work with either representation.

1.5. Absolute and Relative URIs

Since absolute and relative URIs are commonly used in the linking information present in markup language documents, the following is a brief description the two types of URIs. Generally, an absolute URI is a full and complete address for identifying a document or resource in a network, such as an intranet or the public Internet. A relative URI is only a part of an address which extends a given absolute address with additional qualifications.

First, in the case of the HTML markup language, specifically in version 3.2, consider an exemplary home page for Organization-**2** having the URI of http://www.org2.org/. Suppose that another directory, TR/, holds all technical reports of Organization-**2**. Then, with HTML 3.2 a relative link from the home page to the TR/directory can be represented by the following linking information:

```
<a href="TR/">Technical Reports</a>
```

When a user activates this linking information, represented by the words "Technical Reports" displayed on the homepage with characteristic formatting, the linking information is resolved to the absolute URI address, namely http://www.w3.org/TR/. This resolution is accomplished by concatenating the contents of the relative linking information to the absolute address of the document displaying the relative linking information.

The HTML 3.2 anchor tag also allows a link to point to specific sections, or locations, within a document. These sections or locations are known as document fragments. For example suppose the file home.html at URI address http://www.org2.org/home.html has five sections, and a link specifically to Section 4, and not to the start, or top, of home.html is desired. First, Section 4 is specifically identified in the document by placing an anchor at the start Section 4 with, for example, the following syntax:

```
<a NAME="section4"><h2>Section 4</h2></a>
```

Herein, "#section4" is called a fragment identifier. Then, Section 4 in home.html can be specifically and directly addressed in another document with the absolute URI including the fragment identifier:

```
http://www.org2.org/home.html#section4
```

In the home.html document, Section 4 can be specifically and directly addressed with the following relative URI including the fragment identifier:

```
Click here <a href="home.html#section4">Section4</a> to go to
  Section4.
```

With these links, a user does not need to scroll from the start of home.html to get to Section 4.

The methods of this invention do not distinguish between fragments of the same document, because, a reference to the start of a document or to any of its fragments causes download of the complete document. Therefore, a user can scroll to any other part of that document following access of a link to any fragment of that document. Thus, when an absolute URI is resolved from a relative URI in linking information found, fragment identifiers (e.g., #section4) are deleted, or ignored, before comparison to the allowed URI list. If the comparison shows that the URI is allowed, then the fragment identifier part in the filtered documents is used to access the desired location within the document.

Accordingly, the allowed URI list need not maintain representation of allowed fragment identifiers. In other words, access granularity according to this invention is at the level of the document.

The XML markup language also uses absolute and relative linking information, including absolute and relative

URIs. XPointer defined elements can also be used in conjunction with the URI structure, as fragment identifiers or queries, to specify a more precise location inside a document.

2. Browser Functionality

This invention provides in a client application context and within the client application window a browser sub-window with browser functionality configurable in response to user profile parameters. Relevant user profile parameters are obtained for the client application when the user is first authenticated for client-server system access. In particular, the allowed URIs that can be accessed by a user are controlled by the user profile parameters. The client application interface itself is also preferably personalized according to the preferences and assignments of each user.

The browser sub-window inside the application window preferably has a basic subset of the browser functionalities present in a standard markup language document browser program. For example, it is advantageous that at least basic navigation functions, represented by icons such as back, forward, reload, stop, and home, are allowed for all users. Basic search and bookmark functionality is also preferable. However, other features of browsers are preferably enabled or disabled for different categories of users. These configurable features include support for markup language forms, tables, applets, frames, and so forth, document printing, server cookies, task multithreading, and so forth. Generally, browser functionality can be configured in several manners in order to meet the requirements of several categories of users.

Importantly, the browser functionality of this invention also has support for limiting allowed linking information, or URIs, that can be accessed by a user. The browser functionality achieves this limitation by preventing a user from ever sending a request to a HTTP server that is not allowed. In a preferred embodiment, the browser functionality both removes any editable field for entry of URIs to be accessed, and also removes unauthorized linking information from markup language documents before their display. To remove unauthorized linking information before display, all markup language documents are intercepted, their contents parsed, and the parsed contents filtered to find and delete linking information that is indicated as not allowed in the user profile.

For example, in the exemplary CPR system, a physician has relatively unrestricted network access, and the browser sub-window presents both an editable location entry field and bypasses filtering of markup language documents. On the other hand, a patient has relatively restricted access, and any location field is either removed or made not editable for entry of URIs, and linking information in markup language documents is filtered in view of the user profile.

In the following subsections, details of the application interface and its use are presented, first, for the case of unrestricted network access, and, second, for the case of restricted network access. The last subsection presents a preferred implementation of the application program and embedded browser functionality.

2.1. Unrestricted Browser Interface

The graphical user interface (GUI) of a client application with embedded browser functionality of this invention can take a wide variety of layouts and arrangement, each adapted to the needs of the particular client application and the preferences of users. In a preferred embodiment independent of the layout and arrangement, the client application typically displays, according to a windowing paradigm, application specific data and application specific control, and also, when the browser functionality is activated, also browser specific data and browser specific controls. Application specific data is, of course, the substantive data relevant to the application, for example, medical records information for the exemplary CPR application. Application specific controls provide facilities for controlling application functioning, and, also in the preferred embodiment, include display regions selectable with a pointing device or a keyboard in order to activate particular application functions. Such display regions can include menu bars, tool bars, icons, buttons, dialog boxes, and so forth as known to one of ordinary skill in the art. Browser specific data typically includes markup language documents formatted and displayed according to their included markup instructions. Browser specific control are, similar to the application specific controls, activated in order to control browser functioning.

The display of these controls and data can be laid out and arranged in many manners in different windowing paradigms. In preferred embodiments, the application specific controls and data will be spatially separated and displayed in a main application window. The browser specific controls and data will be similarly spatially separated in a browser sub-window. These various display elements can be resized, rearranged, temporarily hidden, and so forth; windows can be tiled, cascaded, overlapped, minimized, maximized, and so forth; all as is known in the art. This invention encompasses all these well-known windowing layouts and arraignments.

Without limitation, FIG. **3** illustrates one exemplary layout and arrangement of the GUI of this invention appropriate for simultaneous display of the display elements of the preferred GUI. In this figure, window **30** is the client application main window configured for a less restricted user, such as a physician. Horizontal toolbar **31** preferably displays application specific controls that are generic to most application sub-functions. Illustrated in this toolbar are, for example, controls for Patient-List display, Tools selection, Help request, and browser toggle **37**. The Browser toggle control activates and deactivates the embedded browser functionality. Vertical toolbar **32** preferably displays further application specific controls that are specific to a current application sub-function. Illustrated here are controls requesting various sections of the CPR of a patient already selected from a previously displayed patient list. Contents of the requested CPR sections, the application specific data in this example, are then displayed in report **33**, which preferably can be adjusted in size.

In FIG. **3**, the browser functionality is activated and browser sub-window **34** is intermediately sized within the application window. The browser sub-window includes toolbar **35**, displaying browser specific controls, and sub-window **36**, displaying browser specific data, namely a retrieved markup language document. Because the physician user here has unrestricted network access, the browser specific controls include editable location entry field **38** for the free entry of URIs to access.

This windowing interface is accessed according to the following steps, which result in the information display illustrated in FIG. **3**.

1. (login) Upon first invoking the CPR application program at an end-user device, a login screen appears where the user enters authenticating information, for example a user ID and password, or presents biometric information, such as a scanned fingerprint.
2. (authentication) The application then authenticates the user and the user's access rights by, for example,

invoking authentication methods on tier **2** servers which access information in the user directory and compare it to entered authenticating information. After authentication, user profile records are loaded to the memory of the end-user device.

3. (application interaction) The application then displays application window **30** configured according to directions in the loaded user profile records and the user commences application interaction with the application specific controls. For example, the user then obtains a patient list by activating the Patient-List control, selects a patient from the list, and then selects a section of CPR of that patient by activating the desired report control. The contents of the desired section are then displayed in report sub-window **33**.
4. (browser activation) When network access is needed in the context of the client application, the user starts the embedded browser functionality by activating browser toggle control **37**. For example, this control displays "Open Browser" when the browser is inactive, and "Close Browser" when active.
5. (browser interaction) The activated browser functionality displays resizeable browser sub-window **34**, including browser specific controls **35** and sub-window **36** for browser specific data. Because the user profile records indicate unrestricted access, the browser specific controls include editable location entry field **38**, and the markup language documents displayed in sub-window **36**, for example, a document initially displayed, are not filtered, having intact linking information. Accordingly, the user can interact with the application browser functionality in a substantially standard manner by entering URI address in location field **38**, or activating linking information in displayed documents.
6. (logoff) When finished, the user logs out of the application, which closes application main window **30**.

2.2. Restricted Browser Interface

Importantly, according to the present invention, the specific controls and data are actually displayed in the application window in a manner responsive to a user's authorities and preferences indicated that user's profile records. For example, access to certain application functions can be prevented for unauthorized users by not displaying the controls by which they can be activated. Browser functionality is similarly configured in a preferred embodiment according to a user's indicated authorizations by displaying only controls for those functions for which that user is authorized. If a user's network access is restricted, the browser sub-window does not display an editable location entry field and unauthorized linking information is filtered from markup language documents before display.

For the exemplary CPR application, FIG. **4** illustrates exemplary application main window **40**, which is similar to main window **30** of FIG. **3** except that it is configured for a user of more restricted authority and access, such as a patient. This window, like that in FIG. **3**, is not limiting, but is exemplary of one layout and arrangement of the GUI of the preferred embodiment of this invention. In FIG. **4**, in contrast with application window **30** of FIG. **3**, generic application toolbar **41** does not display the Patient-List control, since a patient is permitted to view only that patient's CPR. Also, CPR-display sub-function toolbar **42** does not display the Message control, since messages between care providers are considered to be private.

The network access of this user being also indicated as restricted in the user's profile records, accordingly, a location field is not present in the browser specific control toolbar included in browser sub-window **43**. Alternately, a current location field may be displayed that is disabled for entry of linking information. Also since markup language documents are filtered before display, only allowed linking information, as indicated in the user profile, is displayed for user activation in markup language documents.

The steps for accessing the user interface and displayed information illustrated in FIG. **4** are the similar as those described with reference to FIG. **3**, namely the steps of login, authentication, application interaction, browser activation, browser interaction, and logoff. Any differences are clear limitations due to the greater access restrictions placed on this user.

2.2.1 Home Page for Restricted Users

In an alternative embodiment, network access for restricted users makes explicit use of the allowed URIs represented in the user profile records. In one such alternative, the browser presents specific controls enabling user selection of any allowed URIs. Such specific controls can be a drop-down selection list structured either as hierarchical tree-structured bookmarks representing the allowed URIs, or more simply menus listing all allowed URIs for selection. In another alternative, the browser dynamically creates and displays, either upon initial activation or upon user request, a personalized markup language document, a user "home page", which includes linking information representing all allowed URIs, so that the user may access a resource by activating its linking information. This home page can be formatted according to either HTML or XML syntax.

In this alternative embodiment, both the URI itself along with text descriptive of the addressed resource can be presented to the user. Preferably, the descriptive text is the title of the addressed document. A "crawler" program, as known in the art, can periodically access each allowed URI address on the network to retrieve the document title, or other descriptive text, and to store it along with the allowed URI. Additionally, the crawler program can also check if allowed URIs are current and reachable.

This alternative is further described with reference to FIG. **2**. If the user's only allowed URIs are those for node J and all its children, a dynamically created home page need only display linking information representative of the URI http://www.org3.com, along with optional descriptive text. If the user can access all the allowed nodes of FIG. **2**, then the list of allowed URIs has seven parent URI entries, and a dynamically created home page includes linking information representative of these seven URIs, along with optional descriptive text.

The browser functionality of the application is modified to include instructions to access the profile of a restricted user and its stored allowed URI list to dynamically create such a home page or browser specific controls.

2.3. Application/Browser Implementation

The application, and its browser functionality, are implemented with encoding instructions and data placed in end-user device memory that together cause the end-user device processor, and the other end-user device components, to function according to this invention. Such instructions and data resident in an end-user device are generally the means by which the methods and actions of the present invention are accomplished. In particular, the necessary data relevant to this invention derives from the content of the user profile records and, additionally, of the user dictionary records. This content specifies user personalization, authorizations, allowed URIs, and so forth. The necessary instructions

include instructions encoding application function which format application specific controls in a manner responsive to the user profile data as described. They further include instructions encoding browser functionality responsive, in the manners described, to the content of the user profile records.

The application instruction can first be implemented in any convenient programming language and then extended with the browser functionality of this invention in standard manners known to one of skill in the art. Preferred languages are object oriented in order that server objects, present in the preferred client-server architecture described, can be directly invoked. More preferred is use of the Java language, so that the resulting instructions are platform independent and executable in any Java-compatible environments, such as operating systems or markup language browsers with Java virtual machines. Such Java coded applications can be installed either as a standalone application or downloaded as an applet specified in a markup language document.

In particular, the graphical user interface components of the application and its embedded browser functionality can be preferably specified with the Swing component set of the Java Foundation Classes (JFC) from Sun Microsystems. The Swing component set assists in specifying such windowing components as menus, tool bars, dialogs, and so forth, and extends the basic Abstract Windows Toolkit (AWT) of Java. This component set provides classes and application programming interfaces (API), for example, an HTML Editor Kit, the JEditorPane class, the JPanel class, and so forth, that can be directly used to implement the described browser functionality. See, e.g., The Swing Connection, http://www.javasoft.com/products/jfc/tsc/index.html).

Additionally, software components already providing complete portions of standard browser functionality can be imported into a client application and modified according to the present invention. For example, Sun's JavaHelp components use the Swing and other Java libraries to parse and display HTML 3.2 help documents. See, e.g., JavaHelp; a New Standard for Application Help and Online Documentation, http://www.javasoft.com/products/javahelp/). ICEsoft's ICE Browser Java component can be embedded in applications to provide standard browsing functions in an application sub-window. These standard functions can then be extended and modified to provide the browser functionality of this invention. See, e.g., ICEsoft A/S, Bergen Norway, http://www.icesoft.no/products.html). Alternatively, the HotJava browser bean can be embedded and similarly modified and extended. This is a less preferable alternative because HotJava has a large initial size which would grow with needed modifications.

Optionally, people with special needs can be accommodated with accessibility extensions to the browser functionality which are advantageously based on Sun Microsystems' Accessibility API (Accessibility: Support for People with Disabilities, http://www.javasoft.com/products/jfc/index.html#access). This API provides an interface by which technologies for meeting special needs can interact and communicate with Swing and Abstract Windows Toolkit (AWT) components. This API provides the necessary support for accessibility technologies to locate and query user interface objects inside a Java application running in a Java virtual machine. Thus, it is possible to design the embedded browser functionality so that it interfaces with a speech recognition engine. The browser can also use the Accessibility API to interface with screen readers.

3. Filtering Markup Language Documents

As set forth above, restricting user network access to only allowed URIs is achieved, in part, by filtering linking information that is not indicated as allowed in the user profile from displayed markup language documents. In this section filtering methods for HTML and XML documents are described. The type of a markup language document can be easily determined from document type declarations in its header information. These filtering algorithms are preferably implemented to be tolerant of common markup language syntax errors including, at least, missing end tags.

These algorithms involve parsing and string matching, and are advantageously implemented as separate modules in languages specialized for performing string matching, such as sed or perl.

Alternatively, they can be implemented integrally with the browser functionality, preferably in Java.

3.1. Filtering HTML Documents

FIG. **5** illustrates the preferred method for filtering HTML documents. This invention also encompasses alternatives apparent to those of skill in the art in view of this figure and its accompanying description.

First, embedded browser functionality **50** requests an HTML document from allowed HTML server **51**. Upon return of the requested document, the contents of the user profile are checked at step **52** to determine if the user has unrestricted network access. If so, filtering is bypassed for the returned HTML document. If not, the users network access is restricted, and the returned document is filtered to delete linking information that is not allowed.

Step **53**, the first step in HTML document filtering, parses the returned document in order to locate all linking information, or linking elements. These elements are simply identified as those whose start-tag contains an href attribute identifying an addressed resource, such as a markup language document or part of a markup language document. Accordingly, this steps searches for all occurrences of the href attribute, which are recognized by the letters "href", followed by zero or more spaces, followed by the "=" character, followed by zero or more spaces, and finally followed by the '"' character. All linking elements found are stored in a list for subsequent processing.

Each link element found and stored in the list is selected for processing at step **55** until the list is found to be empty at step **54**. When the list is empty, filtering is complete and the markup language document is returned for further processing by embedded browser functionality **50**.

Step **56** processes each link element. First, as previously described, each href attribute value is resolved to an absolute address, if the address is initially relative, by using the known absolute address of the current page in which the link element was found. Next, the representation of the allowed URIs from the user profile is searched for this resolved absolute address. The search ignores any fragment identifier part that may be present in the resolved absolute address. If no allowed URI in the list matches the absolute address, then the link is not allowed, and it is filtered out by deleting the start-tag and the corresponding end-tag of the element containing the href attribute from the HTML document. Examples of deleting links that are not allowed are presented in Appendix A.

Parsing and string matching are performed in a case-independent manner since URIs and HTML tags are case-insensitive.

3.2. XLink Overview

Before describing the method of filtering XML documents, a brief summary of XML linking information, as defined by the XLink specification, is provided. One of skill in art will immediately appreciate how to adapt the methods presented to any future changes to the XLink specification.

XLink linking elements are identified by the presence of the reserved xml:link attribute. Each linking element has one or more associated locators that identify the addresses of the remote resources participating in the link. In a locator the href attribute identifies the address of a resource, either as an absolute URI, or as relative address, i.e., a fragment identifier (XPointer) relative to an absolute address, or both.

XML linking elements are either inline or out-of-line, and also either simple or extended. In an inline link, the content of the linking element includes the associated locator serving to identify the address of the participating resource. A simple inline link has only one locator and combines into a single element the functions of a linking element, a locator, and a descriptor of resource attributes. The HTML anchor element, <a . . . >, is an example of a simple inline link. Another example of an inline simple link is shown below:

```
<P>The headquarters of <mySimpleLink xml:link="simple"
  inline="true" href="http://www.org3.com">organization-3</
  mySimpleLink> are moving to Amsterdam.</P>
```

Here, mySimpleLink is a user-defined simple linking element. Inline links can also be extended, containing multiple locators addressing multiple resources.

An out-of-line link, in contrast, does not itself identify the link's addressed resources, instead requiring one or more associated, but separate, child, or dependent, addressing elements that specify the addresses of linked resources. An extended out-of-line link addresses many resources, separating the identification of a link from the identification of the addressed resources. The following is an example of an extended out-of-line link:

```
<myExtendedLink xml:link="extended" inline="false">

<locator href="products.xml"/>

<locator href="service.xml"/>

<locator href="http://www.org4.com"/>

</myExtendedLink>
```

Here, myExtendedLink is a user-defined extended out-of-line linking element addressing three resources, two expressed relative to the document in which the linking element is contained and the third expressed as an absolute address. One way that an extended out-of-line link can be used is that the user is provided with a menu list of links (for example, three in the example above) and can choose which one to go to. Simple out-of-line links addressing only one linked resource are also possible.

Dependent addressing elements can be "locator" elements, as in the above example. Dependent addressing elements can also be a "group" of "document" elements, forming together an interlinked group. The "group" elements marking an extended link group which is a collection of "document" elements that contain links to other documents, Each linked document in the group is identified by a associated "document" element. "Document" elements therefore also have href attributes specifying the addresses of linked resources.

3.3. Filtering XML Documents

FIG. **6** illustrates the preferred method for filtering XML documents. HTML and XML documents can be distinguished by the document type declaration appearing at the beginning of each markup language document. This invention also encompasses alternatives apparent to those of skill in the art in view of this figure and its accompanying description.

First, embedded browser functionality **60** requests an XML document from allowed XML server **61**. Upon return of the requested document, the contents of the user profile are checked at step **62** to determine it the user has unrestricted network access. If so, filtering is bypassed for the returned XML document. If not, the user's network access is restricted, and the returned document is filtered to delete linking information that is not allowed.

Step **63**, the first step in HTML document filtering, parses the returned document in order to locate all linking information, or linking elements, and to store them in a list for subsequent processing. Linking elements are indicated by the xml:link attribute. This attribute may be explicitly present in a linking element start-tag, as illustrated in the above examples, or it may be indirectly indicated as a default value in an attribute-list declaration of a linking element. For example, the attribute-list declarations

```
<!ATTLIST mylink xml:link CDATA #FIXED "simple">
```

declares that all "mylink" elements in the current XML document are simple linking elements. Accordingly, all attribute-list declarations as well as element start-tags are parsed in order to recognize all linking elements.

Each linking element found and stored in the list is selected for processing at step **65** until the list is found to be empty at step **64**. When the list is empty, filtering is complete and the markup language document is returned for further processing by browser functionality **60**.

Next, step **66** checks the inline attribute of each linking element to determine if the linking element is inline or out-of-line. The default value for the inline attribute is true. For inline linking elements, step **67** finds, for simple inline links, the href attribute present, or, for extended inline links, the href attributes present. For each found href attribute, step **68** resolves, if necessary, the address indicated in the href attribute to an absolute URI address, searches the representation of the allowed URIs for the indicated absolute address (minus any fragment identifier XPointer part), and deletes the href attribute from the element start tag if it is not allowed. Steps **69** and **70** delete the linking element start tag and corresponding end tag from the document if all the contained href elements are not allowed and have been deleted from the start tag.

Processing out-of-line linking elements is similar except that all child, or dependent, addressing elements which specify the addresses of linked resources must first be found. Accordingly, step **71** finds all such locator, group, and document elements. For locator elements, step **72** finds its href attribute, resolves this attribute to the absolute address if necessary, checks the list of allowed URIs for the resolved absolute address, and deletes the locator element if the href attribute is not in the allowed URI list. Step **73** and **74** delete the out-of-line linking element's start tag and corresponding end tag from the document if no locator elements remain in the start tag.

The processing of group and document addressing elements is similar. Accordingly, step **71** finds all such locator and document elements and notes the value of their href attribute. For locator elements step **72** resolves the href attribute to the absolute address if necessary, checks the list of allowed URIs for the resolved absolute address (minus any fragment identifier part), and deletes the corresponding locator element if the href attribute is not in the allowed URI list. Step **73** and **74** delete the out-of-line linking element's start tag and corresponding end tag from the document if no child locator elements remain for the out-of-line element.

The processing of extended link groups is similar. Group elements are found in step **63** by checking for xml:link

attribute values of "group". Step **71** finds all document elements of the extended link group and notes the value of their href attribute. For document elements step **72** resolves the href attribute to the absolute address if necessary, checks the list of allowed URIs for the resolved absolute address (minus any fragment identifier part), and deletes the corresponding document element if the href attribute is not in the allowed URI list. Step **73** and **74** delete the group element's start tag and corresponding end tag if no child document elements remain for the extended link group element.

3.4. Extension to Other Domains

The filtering methods described in this section, and this invention generally, are not limited to only HTML and XML documents with linking information formatted as URIs, but can be generally applied to any markup language documents with a syntax providing identifiable markup elements serving as linking elements indicating network addresses of accessible resources. One of skill in the art, in view of the previous description, will appreciate how to store lists of linking information in other formats, possibly compressed if appropriate, and how to modify the parsing steps in the described filtering methods to recognize linking element defined by other markup languages.

In particular, these methods can be applied to markup languages being developed by the Advanced Television Systems Committee (ATSC) for the consumer electronics industry. See, e.g., Broadcast HTML Specification, http://toocan.philabs.research.philips.com/misc/atcs/bhtml; Wugofski, 1998, a Modular Hypertext Markup Language for Broadcast Applications, Draft no. 4, Over the Moon Productions/Gateway. The ATSC is in the process of developing application programming interfaces for a Digital Television Application Software Environment (DASE) compliant receiver, including markup languages which are similar to HTML.

For example, xHTML specifies a collection of document type definition (DTD) sets that can be combined to specify xHTML-based platforms, such as the w3HTML, bHTML, and cHTML platforms. The w**3**HTML platform provides support for full World Wide Web (WWW) connectivity. The cHTML platform provides a compact profile for appliance-oriented products. The bHTML platform provides a profile for TV-centric products written in XML, including HTML elements and attributes, and including new synchronization functionality as new elements, attributes, and style properties.

All these markup languages define linking elements with an href attribute indicating a URI specifying the address of a network resource. These linking elements define links between a current element in a current document and the destination anchor in a destination document.

In addition to accommodating the TV-type appliance applications above, the methods of this invention can also be generally applied in other consumer appliances. Generally, it is anticipated that browser functionality will become available in standard consumer produces, for example, microwave ovens and other kitchen appliances, pagers and other specialized information appliances, and so forth. These appliances will be able to connect to Internet servers for product upgrades, feature upgrades, bug fixes, and so forth. This embedded browser functionality can advantageously use the methods and apparatus of this invention, along with a permanently stored list of allowed URIs, to limit Internet access of the consumer appliances to the specified portal servers, problem fix servers, upgrade servers, or so forth. Accordingly, these specific servers can be accessed for their specific functions by a single button and with no consumer skill required.

4. Running an Applet Versus an Application

If the application, as well as its embedded browser functionality, is coded in the Java language, it can be installed in an end-user device either as a standalone application or as an applet downloaded and executed inside most commercially available markup language browser programs. Installation as an applet, instead of an application, is advantageous in that the applet can be upgraded by merely placing a new version on a server machine, whereas application upgrade requires placing the new version on every end-user device. Further, applets provide easy installation for most end-user devices because browser programs are, or will be, widely available for almost all end-user devices. Applets do have the disadvantage of requiring download time.

Use of a Java runtime plug-in permits a standard Java virtual machine environment for applets across the range of commercial markup language browsers. Such plug-in technology is available from Sun Microsystems' (Java Plug-In, http://www.javasoft.com/products/plugin/). Further, applets can be marked as "signed" so that they are trusted to access local files and to open connections to network servers other than the one from which they were downloaded. Additionally security can be provided by local configuration code, which checks policy files, and, if appropriate, authorizes individual applet access to files or network addresses.

FIG. **7** illustrates an exemplary and non-limiting arrangement and layout of the graphical user interface of this invention as presented by a signed applet being run by a commercial markup language browser. Here, window **80** is the commercial browser's standard window with its own toolbar **81**. Inside this window, sub-window **82** presents the display generated by the applet of this invention. Here, similarly to FIG. **3**, this is illustrated as the display for a less restricted physician-type user. Of course, for a more restricted patient-type user the display would be more similar to FIG. **4**. Browser toggle control **83** on the top right-hand corner of the applet sub-window activates or deactivates the browser functionality embedded in the applet application. Accordingly, browser sub-window **84** is in effect a browser in a browser.

Use of an applet differs from use of an application only in that the user initially starts a commercial browser available on the end-user device and directs it to the URI of the HTML page referencing the signed Java applet. Such an applet reference is identified with an applet tag having a code attribute identifying the name of the applet class filename, and an optional "archive" attribute identifying a jar file in which the class is archived.

Appendix A

This appendix presents examples of deleting HTML links that are not allowed. In HTML 3.2 the anchor tag <a . . . > and the area tag <area . . . > are usually used with href attributes to specify linking information. For example, consider the following simple HTML source segment that uses the anchor tag:

```
<P>The headquarters of <a href="http:
  //www.org3.com">Organization-3</a> are moving to Amster-
  dam.</P>
```

A markup language browser will typically display this as:

The headquarters of Organization- 3 are moving to Amsterdam.

Clicking on the underlined word will direct the browser to download the document at the address http://

www.org3.com. If this document is not allowed for a user, the HTML filtering methods, upon parsing the document, detect the start-tag containing the linking information 'href= "http://www.org3.com"' and the corresponding end-tag. Since this is not in the allowed URI list, these start and end tags are deleted, leaving the following in the filtered document:

```
<P>The headquarters of Organization-3 are moving to Amsterdam.</P>
```

Because the anchor tag is deleted, no link is displayed to the address that is not allowed.

Image maps are another mechanism for specifying linking information. Consider the following HTML document fragment:

```
<map name="bottom">
<area shape="rect" alt="Organization-3 Home Page"
coords="1, 1, 100, 50"
href="http://www.org3.com/"></area>
<area shape="rect" alt="Welcome Page"
coords="200, 300, 250, 350" href="/welcome.html"></area>
</map>
<a href="index.map"><img src="index.gif"
ISMAP="#bottom"></a>
```

Here a browser would display the image file, index.gif, with two hotspot rectangles, one in the upper left-hand corner and the other in the lower-right hand corner at the specified coordinates. The upper hotspot rectangle is a link to the Organization-3 home page, while the lower hotspot links to the welcome page in the same relative directory as the document fragment. If the Organization-3 home page is not allowed for the user, then the HTML filtering methods will detect and delete the area start-tag containing the attribute href="http://www.org3.com/" and the corresponding end-tag. The filtered document becomes:

```
<map name="bottom">
<area shape="rect" alt="Welcome Page"
coords="200, 300, 250, 350" href="/welcome.html"></area>
</map>
<a href="index.map"><img src="index.gif" border=0
ISMAP="#bottom"></a>
```

Display of this filtered HTML document presents only the lower hotspot linking to the welcome page in the same relative directory as the document fragment.

All references cited herein are incorporated herein by reference in their entirety and for all purposes to the same extent as if each individual publication or patent or patent application was specifically and individually indicated to be incorporated by reference in its entirety for all purposes.

What is claimed is:

1. An apparatus for a user to browse markup language documents, the documents being stored for retrieval at one or more computer systems attached to a network and containing linking information representing addresses of network resources, the apparatus comprising:

an end-user device comprising processor, memory, network attachment, input means, and display, wherein the memory stores user profile records whose contents describe characteristics and preferences of the user relating to the user's restrictions to at least one of access to network resources and use of browser specific controls, and

an application program comprising:

(i) instructions for causing the processor to perform an application function which displays an application window including application specific controls and data, wherein the application specific controls are responsive to the content of the user profile records and comprise a browser control, and

(ii) instructions for causing the processor to perform, responsive to user activation of the browser control, a browser function which displays within the application window a browser sub-window including browser specific controls and markup language documents, wherein both the displayed browser specific controls and linking information retained in the displayed markup language documents are responsive to the content of the user profile records.

2. The apparatus of claim **1** wherein the browser specific controls comprise a location entry field for user input of linking information representing resource addresses to be accessed, and wherein the browser function displays or not the location entry field responsively to contents of the user profile records.

3. The apparatus of claim **1** wherein contents of the user profile records comprise an indication of allowed linking information, and wherein the browser function retains only allowed linking information in the displayed markup language documents, whereby the user can only access only allowed resources from a displayed markup language document.

4. The apparatus of claim **3** wherein the browser function filters markup language documents in order to delete linking information that is not indicated as allowed by contents of the user profile records.

5. The apparatus of claim **4** wherein the markup language documents comprise HTML documents, and wherein the browser function filters an HTML document by parsing the document to find href attributes, by locating linking information in found href attributes, by resolving addresses in located linking information to absolute addresses, and by deleting located linking information from the HTML document that is not indicated as allowed by contents of the user profile records.

6. The apparatus of claim **4** wherein the markup language documents comprise XML documents, and wherein the browser function filters an XML document by parsing the document for XLink elements, by locating linking information from all href attributes associated with found XLink elements, by resolving addresses in located linking information to full address, and by deleting located linking information from the XML document that is not indicated as allowed by contents of the user profile records.

7. The apparatus of claim **6** wherein the href attributes associated with an XLink element are either (i), if the XLink element has a simple type, located in the element start-tag, or (ii), if the XLink element has an extended type, located in locator and document element children of the XLink element.

8. The apparatus of claim **3** wherein the indication of allowed linking information indicates only whether or not entire markup language documents are allowed.

9. The apparatus of claim 1 wherein the memory also stores instructions for interpreting Java codes, and wherein the instructions of the application program comprise interpretable Java codes.

10. The apparatus of claim 1 wherein user profile or user directory databases are resident on one or more network attached computer systems, and wherein the application program further comprises instructions for downloading the user profile records from the user profile or user directory databases.

11. The apparatus of claim 1 further comprising business data server systems which provide business data requested by the end-user device.

12. The apparatus of claim 1, wherein the user profile records relate to the user's restrictions to access to network resources and use of browser specific controls.

13. The apparatus of claim 1, wherein said user profile records are created for each user but not by each user so that each user does not have access to his or her user profile record and cannot alter his or her user profile record.

14. A method for a user to browse markup language documents at an end-user device, the documents being stored for retrieval at one or more computer systems attached to a network and containing linking information representing resource addresses, the method comprising the steps of:

entering user identification information,

authenticating entered user identification information and loading user profile records associated with the authenticated user to the end-user device, the user profile records relating to the user's restrictions to at least one of access to network resources and use of browser specific controls,

displaying an application window at the end-user device including application specific controls and data, wherein the application specific controls are responsive to the content of the user profile records and comprise a browser control, and

displaying, within the application window and responsively to user activation of the browser control, a browser sub-window including browser specific controls and markup language documents, wherein both the displayed browser specific controls and linking information retained in the displayed markup language documents are responsive to the content of the user profile records.

15. The method of claim 14 wherein the browser specific controls comprise a location entry field for user input of linking information representing resource addresses to be accessed, and wherein displaying or not of the location entry field in the browser sub-window is responsive to contents of the user profile records.

16. The method of claim 14 wherein contents of the user profile records comprise an indication of allowed linking information, and wherein displaying markup language documents in the browser sub-window retains only allowed linking information, whereby the user can access only allowed resources from displayed markup language documents.

17. The method of claim 16 wherein all linking information is indicated as allowed.

18. The method of claim 16 wherein displaying markup language documents further comprises filtering markup language documents before display to delete linking information that is not indicated as allowed by contents of the user profile records.

19. The method of claim 18 wherein the markup language documents comprise HTML documents, and wherein the filtering HTML markup language documents further comprises:

parsing an HTML document to find href attributes,

locating linking information from the found href attributes,

resolving addresses in located linking information to absolute addresses, and

deleting located linking information that is not indicated as allowed by contents of the user profile records.

20. The method of claim 18 wherein the markup language documents comprise XML documents, and wherein the filtering XML markup language documents further comprises:

parsing an XML document to find XLink elements,

locating linking information from all href attributes associated with found XLink elements,

resolving addresses in located linking information to absolute addresses, and

deleting located linking information that is not indicated as allowed by contents of the user profile records.

21. The method of claim 20 wherein the locating linking information further comprises finding all href attributes associated with a found XLink element by

if the XLink element has a simple type, finding all href attributes in the element start-tag, and

if the XLink element has an extended type, finding all href attributes in locator and document element children of the XLink element.

22. The method of claim 16 wherein the displaying a browser sub-window further comprises:

creating dynamically a homepage markup language document which includes the linking information indicated to be allowed, and

displaying the homepage markup language document in the browser sub-window so that the allowed displayed linking information can be accessed by the user.

23. The method of claim 16 wherein the browser specific controls further comprise a control the activation of which causes display of a selection list presenting the linking information indicated to be allowed for user selection and access.

24. The method of claim 16 wherein the indication of allowed linking information indicates only whether or not entire markup language documents are allowed.

25. The method of claim 14 wherein user profile or user directory databases are resident on one or more network attached computer systems, and wherein loading the user profile records to the end-user device comprises downloading the user profile records across the network from the user profile or user directory databases.

26. The method of claim 14 further comprising, prior to the step of entering identification information, a step of directing a browser program active in the end-user device to a markup language document which causes loading of instructions into the end-user device for performing the subsequent steps of displaying an application window and displaying a browser sub-window.

27. The method of claim 14, wherein the user profile records relate to the user's restrictions to access to network resources and use of browser specific controls.

28. The method of claim 14, further comprising the steps of:

creating for each user but not by each user a user profile record, and

preventing access by each user to his or her user profile record so that each user cannot alter his or her user profile record.

* * * * *